الفريق العربي للهندسة العكسية

Arab Team Reverse Engineering

فاتح رمضان

1430



2nd Edition

AT4RE

الفهرس

إهداء ...

إلى من ستنجب ذلك الرجل الذي سيرفع رايتي لا إله إلا الله محمد رسول الله فوق سماء القدس..

إلى من نفضوا عنهم غبار التسويف والتواكل والأماني ليجعلوا من مبادئهم ومعتقداتهم وأقوالهم أفعالاً تنهض بهم وبأمتهم..

إلى كل من ضحّى ويضحي بوقته وجهده من أجل إيصال علم الهندسة العكسية للعرب والمسلمين

إلى كل أعضاءنا الأعزاء في الفريق العربي للهندسة العكسية.

إلى هؤلاء جميعاً نهدي هذا الكتاب، آملين أن يجدوا فيه المتعة والفائدة، سائلين الله عز وجل أن يكون خالصاً لوجهه الكريم..

تحية لكم جميعاً

المقدمة

بسم الله الرحمن الرحيم

﴿وَاللَّهُ أَخْرَجَكُم مِّن بُطُونِ أُمَّهَاتِكُمْ لَا تَعْلَمُونَ شَيْئًا وَجَعَلَ لَكُمُ السَّمْعَ وَالْأَبْصَارَ وَالْأَفْئِدَةَ لَعَلَّكُمْ تَشْكُرُونَ﴾

[النحل : 78]

الحمد لله المبدئ المعيد يبدأ الخلق ثم يعيده وهو أهون عليه وله المثل الأعلى، لا يَعزُب عنه مثقال ذرةٍ في السموات ولا في الأرض ولا أصغر من ذلك إلا بعلمه لقد أحصاهم وعدهم عداً، هل تحس منهم من أحدٍ أو تسمع لهم رِكزاً، خشعت الأصوات للرحمن فلا تسمع إلا همساً .. والصلاة والسلام على سيدنا محمد وعلى آله وصحبه وسلم، وبعد:

منذ أن تم انطلاق موقع الفريق العربي للهندسة العكسية عام 2007 ونحن نسعى بجدٍ ونشاطٍ إلى إيصال علم الهندسة العكسية للعرب بمجالاته الواسعة التي كانت حكراً للغرب فقط.

لقد كانت بدايتنا متواضعة ومتعبة جداً، وخاصة وأنّ مفردات هذا العلم جديدة وما نشر عنه من كتبٍ ومواضيعٍ باللغة العربية يكاد أن يكون شبه معدوم ولا يفي بالغرض.

لكن هذا لم يقف عثرةً أمام عزيمتنا وإصرارنا في إيصال هذا العلم، فبدأنا بتقديم الدروس المكتوبة والمصورة بلغتنا العربية، وقمنا بشرح الكثير من أدوات وبرامج هذا العلم، وقدمنا دوراتٍ متخصصة به، ورفعنا من سوية المبرمجين في مجال حماية البرامج عن طريق التمارين والتحديات التي كنا نقدمها ونشارك الأعضاء بها.

لم يقف طموحنا عند هذا الحد، بل أصبحنا ننافس الفرق الأجنبية في هذا العلم لنكون منتجين لا مستهلكين؛ فأصدرنا العديد من الأدوات والبرامج التي صممت بأيدينا وعقولنا، ونطمح اليوم وكل يوم للمزيد المزيد، ودوماً يتحاور أعضاء الفريق حول أفكار وأدوات تخدم هذا العلم ليقوموا ببرمجتها وتقديمها.

ولقد كان الحدث الأكبر في عالم الهندسة العكسية العربي هو إصدار كتابنا الضخم "مدخل إلى الهندسة العكسية" كهدية في رأس السنة الهجرية 1429 حوت ألف باء الهندسة العكسية، وأصبح المرجع الأول لطلبة هذا العلم سواءً أكانوا من المبرمجين أو المتخصصين أو غيرهم من الهواة.

واليوم يغوص بكم الفريق العربي للهندسة العكسية ويسبح معكم في مجال أوسع في عالم الهندسة العكسية عندما يقدم لكم كتابه الثاني [الهندسة العكسية : خطوة إلى الأمام] حيث ستتعرفون من خلال هذا الكتاب على بعض

المفاهيم المتقدمة في علم الهندسة العكسية؛ كخوارزميات التشفير، وفك الحمايات الشائكة، وتحليل الفيروسات، وبرمجة أدوات الفك، وكتابة باتش أو كيجن لبرنامجٍ ما، وتنقيح برامج الدوت نت والجيل الثاني من برامج الجوال، وتصميم سكين للباتشات، والكثير..

وها نحن نتقدم بهذا الكتاب سائلين الله تعالى أن يكون خالصاً لوجهه الكريم، فإن وجدت - أيها القارئُ العزيز - فيه جهداً فادع لنا بالتوفيق والسداد، وإن وجدت زلة قلم فافتح لها باب التجاوز والمغفرة..

فلا بد من عيـــبٍ فإن تجدنه — فسامح وكن بالسّتر أعظمَ مفضل
فمن ذا الذي ما ساء قط، ومن — له المحاسن قد تمت سوى خير مرسل

وأخر دعوانا أنِ الحمد لله رب العالمين...

الفريق العربي للهندسة العكسية

Arab Team 4 Reverse Engineerin

كيف تستعمل الكتاب؟

تم تصفيف الكتاب ليعطي للقارئ والدارس أقصى قدر ممكن من المرونة واليسر أثناء القراءة والتطبيق؛ لذا قد تجد أن الكتاب قد خرج قليلاً عن التبويب المعتاد حيث تم اعتماد تبويب أشمل، كما تم وضع صور رمزيةٍ لإثارة الانتباه أو التحذير على الشكل التالي :

	إثارة الانتباه إلى خطأ ما أو شيء مهم جداً.
	اضغط على هذه الصورة لتحميل المرفقات التابعة لكل درس. وتجدر الاشارة إلى أن **لكل درس مرفق واحد (1)** يحمل اسم الدرس اللاتيني وداخله تجد جميع ما ستحتاجه لمسايرة الدرس.
	للمساعدة.
	فكرة هامة أو ملاحظة هامة.
	خلاصة شاملة.
	سرد الأدوات التي ستحتاجها في الدرس.
	رابط لموقع أو منتدى.

و كذلك تم تعمد عدم التفصيل في بعض الأمور والتي – في اعتقادنا– من الواجب أن تكون على اطّلاعٍ بها؛ لذا نرجو منك أخي الكريم أن ترجع إلى الإصدار الأول من كتابنا (مدخل إلى الهندسة العكسية) عندما يتعذر عليك فهم شيء من أساسيات هذا العلم وقبل طرح أي سؤال، ونحن على ثقة بأنك ستجد فيه بغيتك.

يمكنك تحميل الإصدار الأول من هنا مباشرة أو تفضل بالدخول إلى المشاركة التالية لمزيد من المعلومات .

وقد خصصت مشاركة مستقلة لأسئلتكم، ملاحظاتكم و كذا اقتراحاتكم حول الطبعة الأولى من الكتاب يمكنكم الدخول اليها مباشرة من هنا .

و على نفس النهج خصصنا مشاركة مثبتة لتلقي ملاحظاتكم و تساؤلاتكم المخصصة لطبعتنا الثانية من الكتاب.



الباب الأول: أفكار وحلول متقدمة في OllyDbg

كيف ينقّح OllyDbg البرامج ؟

How Does OllyDbg Debugs Targets

في هذا الدرس سنتعرف بإذن الله على الطريقة التي يعمل بها OllyDbg لتنقيح برنامج ما، وأيضاً الطريقة التي يضع بها نقط توقف من نوع استثناء في التعليمية Int3 .

قبل البدء سنقوم بشرح الدوال المستعملة في الدرس:

- CreateProcess
- GetStartupInfo
- ReadProcessMemory
- WriteProcessMemory
- WaitForDebugEvent
- ContinueDebugEvent
- CloseHandle
- GetOpenFileNameA
- ZeroMemory
- GetThreadContext
- SetThreadContext
- DebugActiveProcess

I. أولاً : شرح الدوال المستعملة في الدرس

1.1 CreateProcess

دالة لتحميل ملف تنفيذي إلى الذاكرة ، من أجل التنفيذ أو التنقيح حسب الباروميتر dwCreationFlags ولهذه الدالة 10 باروميترات كالتالي :

BOOL CreateProcess(	دالة تحميل ملف للذاكرة، وهي ترجع قيمتين: **TRUE** في حالة نجاحها في تحميل البرنامج. **FALSE** إذا فشلت في تحميل الملف. لذلك وجب عمل اختبار قبل الدخول في العمليات على الملف المحمل.
LPCTSTR lpApplicationName	هذا الباروميتر هو مؤشر نحو نص، وهذا الأخير يجب أن يحتوي مسار الملف الذي تريد تحميله للذاكرة من أجل التنفيذ. ملاحظة : في الغالب الملفات من نوع exe أو com هي ما تنفذ أثناء الضغط عليه مرتين.

الآن تخيل أن لديك ملف تنفيذي، لكن بامتداد مختلف؛ مثلاً : jpg أو بدون امتداد حتى وتريد تنفيذه، إذن ماعليك سوى كتابة Loader يحمل ذلك الملف بهذه الدالة.	
مؤشر نحو نص منتهي بـــ 0byte ويحمل الأوامر التي ستنفذ أثناء تحميل الملف.	LPTSTR lpCommandLine
مؤشر نحو بنية من نوع SECURITY_ATTRIBUTES بها بعض خصائص الملف المحمل للذاكرة بالدالة. (لن نحتاجها)	LPSECURITY_ATTRIBUTESlpProcessAttributes
نفس الشيء بالنسبة للبارومتر السابق... (لن نحتاجها)	LPSECURITY_ATTRIBUTES lpThreadAttributes
لن نحتاجه.	BOOL bInheritHandles
هذا هو البارومتر المهم في هذه الدالة، وهو يحتوي على الطريقة التي تحب أن يحمل بها الملف في الذاكرة، وسنذكر بعض الأمثلة. ⊙ CREATE_SUSPENDED : تحميل الملف للذاكرة، ووضعه في حالة انتظار قبل تنفيذ أي تعليمية من أول قسم في البرنامج المحمل. ⊙ DEBUG_PROCESS: تحميل الملف من أجل التنقيح، وسيتم تنقيح أي توابع لهذا البرنامج أيضاً. ⊙ DEBUG_ONLY_THIS_PROCESS : تخصيص التنقيح من أجل الـــ Process الأساسي في البرنامج. أما إن كنت تريد أن ينفذ البرنامج مباشرة بعد تحميله، فيكفي تصفير هذا المؤشر بجعله NULL . في هذا الدرس سنحتاج الجمع بين هاتين الخاصيتين الأخيرتين، وسنجعل هذا البارومتر : **DEBUG_PROCESS | DEBUG_ONLY_THIS_PROCESS**	DWORD dwCreationFlags
لن نحتاجه.	LPVOID lpEnvironment
لن نحتاجه.	LPCTSTR lpCurrentDirectory
مؤشر نحو البنية STARTUPINFO وبنيتها كالتالي : typedef struct _STARTUPINFO { // si DWORD cb; LPTSTR lpReserved; LPTSTR lpDesktop; LPTSTR lpTitle; DWORD dwX; DWORD dwY; DWORD dwXSize; DWORD dwYSize; DWORD dwXCountChars; DWORD dwYCountChars; DWORD dwFillAttribute; DWORD dwFlags; WORD wShowWindow; WORD cbReserved2; LPBYTE lpReserved2;	LPSTARTUPINFO lpStartupInfo

	HANDLE hStdInput; HANDLE hStdOutput; HANDLE hStdError; } STARTUPINFO, *LPSTARTUPINFO; لملء هذه البنية سنحتاج الدالة GetStartupInfo .
LPPROCESS_INFORMATION lpProcessInformation	مؤشر نحو البنية ROCESS_INFORMATION وهي مهمة جداً في جميع تطبيقات هذه الدالة، وبنيتها كالتالي : typedef struct _PROCESS_INFORMATION { // pi HANDLE hProcess; HANDLE hThread; DWORD dwProcessId; DWORD dwThreadId; } PROCESS_INFORMATION; حيث : • hProcess هو مقبض الـ Process ويجب التميز بينه وبين dwProcessId : هو PID للبرنامج، مثل الذي تجده في TaskMan . سنحتاجه من أجل عمل Attach لبرنامج محمل في الذاكرة. لأن المقبض يمكن أن تتغير قيمته باستعمال الدالة OpenProcess لتغيير خصائص التعامل مع البرنامج المحمل للذاكرة. • أما الـ PID فيكون ثابتاً مادام البرنامج محمل للذاكرة. إذ نحن أصلاً سنحمل البرنامج من أجل التنقيح؛ لذلك ستهمنا القيمة hProcess وليس dwProcessId. • hThread : مقبض الـ Thread ويجب التمييز بينه وبين dwThreadId : الـ TID وهي قيمة ثابتة للـ Thread الأساسي في البرنامج. • والفرق بينهم كالفرق بين الاثنين السابقين، لكن بمفهومين مختلفين.
);	انتهت.

GetStartupInfo .1.2

VOID GetStartupInfo(	دالة لملء البنية STARTUPINFO
LPSTARTUPINFO lpStartupInfo	مؤشر نحو البنية STARTUPINFO
);	انتهت.

ReadProcessMemory .1.3

BOOL ReadProcessMemory(	دالة لقراءة بيانات من عنوان ضمن الملف المحمل في الذاكرة .

	ترجع TRUE إن نجحت في التنفيذ.
HANDLE hProcess	مقبض البرنامج المح مل للذاكرة وهو ضمن البنية PROCESS_INFORMATION بعد تنفيذ الدالة CreateProcess .
LPCVOID lpBaseAddress	العنوان الذي تريد القراءة منه، وهو من نوع VA .
LPVOID lpBuffer	المتغير الذي ستحفظ فيه الدالة المعلومات التي قرأته ا من ذلك العنوان.
DWORD nSize	كم عدد البيانات التي تريد قراءتها من العنوان الذي أعطيت الدالة؟
LPDWORD lpNumberOfBytesRead	الدالة تضع فيه عدد البايتات التي قرأت من العنوان . في بعض المرات يكون الحجم الذي أعطينا الدالة أكبر من عدد البايتات في العنوان، وبذلك لن تقرأ ا الحجم بأكمله. وهذا العنوان يحتوي عدد البايتات التي قرأت من العنوان . في الغالب لن نحتاجه، سنجعله NULL
);	انتهت.

1.4 WriteProcessMemory

BOOL WriteProcessMemory(	دالة لكتابة بيانات في عنوان ضمن الملف المحمل في الذاكرة . ترجع TRUE إن نجحت في التنفيذ.
HANDLE hProcess	مقبض البرنامج المحمل للذاكرة، وهو ضمن البنية PROCESS_INFORMATION بعد تنفيذ الدالة CreateProcess .
LPCVOID lpBaseAddress	العنوان الذي تريد القراءة منه، وهو من نوع VA .
LPVOID lpBuffer	المتغير الذي يحتوي البيانات التي تريد كتابتها في العنوان.
DWORD nSize	كم عدد البيانات التي تريد كتابتها في العنوان الذي أعطيت الدالة؟
LPDWORD lpNumberOfBytesWritten	الدالة تضع فيه عدد البايتات التي كتبت في العنوان . في بعض المرات يكون الحجم الذي أعطينا الدالة أكبر من عدد البايتات الممكن كتابتها في العنوان، وبذلك لن تكتب الحجم بأكمله . وهذا العنوان يحتوي عدد البايتات التي كتبت في العنوان . في الغالب لن نحتاجه، سنجعله NULL
);	انتهت

1.5 CloseHandle

BOOL CloseHandle(	دالة لإغلاق مقبض ؛ مثلاً **hProcess** هو مقبض، لذلك يلزمنا إغلاقه . وهي ترجع **TRUE** إذا نجحت
HANDLE hObject	المقبض الذي تريد إغلاقه.
);	انتهت

1.6 ZeroMemory

VOID ZeroMemory(	دالة لتصفير متغير.
PVOID Destination	عنوان المتغير الذي تريد تصفيره.
DWORD Length	حجم المتغير الذي تريد تصفيره.
);	انتهت

(GetThreadContext :

BOOL GetThreadContext(	دالة متميزة وتمكننا من جلب – تقريباً- جميع المعلومات حول الـ **Context** للبرنامج المحمل في الذاكرة. ونقصد بـ **Context** بنية تحتوي كل من المسجلات والرايات **Flags** المعلقة بالملف المحمل في الذاكرة .
HANDLE hThread	هذه الدالة تتعامل مع الـ Threads لذلك وجب إعطاؤها مقبض الـ Thread وهو ضمن البنية PROCESS_INFORMATION بعد تنفيذ الدالة CreateProcess .
LPCONTEXT lpContext	مؤشر نحو البنية CONTEXT وهي كالتالي : typedef struct _CONTEXT { DWORD ContextFlags; DWORD Dr0; DWORD Dr1; DWORD Dr2; DWORD Dr3; DWORD Dr6; DWORD Dr7; FLOATING_SAVE_AREA FloatSave; DWORD SegGs; DWORD SegFs; DWORD SegEs; DWORD SegDs;

	DWORD Edi; DWORD Esi; DWORD Ebx; DWORD Edx; DWORD Ecx; DWORD Eax; DWORD Ebp; DWORD Eip; DWORD SegCs; DWORD EFlags; DWORD Esp; DWORD SegSs; BYTE ExtendedRegisters[MAXIMUM_SUPPORTED_EXTENSION]; } CONTEXT; نظنك فهمت ما نقصد.
);	انتهت.

1.7 SetThreadContext

BOOL SetThreadContext(	دالة لتغيير معلومات سير البرنامج في الذاكرة.
HANDLE hThread	مقبض الـــ Thread الذي تريد التحكم فيه.
CONST CONTEXT *lpContext	مؤشر نحو البنية.
);	انتهت.

مثلاً : لتجاوز قفزة من مشروطة ألا يكفي سوى تغيير قيمة الـ Flag .. مثلاً ZF بالنسبة للقفزتين JZ وJNZ ؟ وهل هذه الـــ Flag توجد ضمن البنية Context ؟ طبعاً نعم، إذن لتغيير هذه الـــ Flag يكفي تغيير البنية Context ثم إعادة طبعها في Thread . يمكنك أيضاً تغيير المسجل Eip لتنفيذ مكان آخر مباشرة.

1.8 GetOpenFileName

BOOL GetOpenFileName(	دالة لفتح OpenDialog لاختيار ملف ما.
LPOPENFILENAME lpofn	مؤشر البنية OPENFILENAME وهي كالتالي : typedef struct tagOFN { // ofn DWORD lStructSize; HWND hwndOwner; HINSTANCE hInstance; LPCTSTR lpstrFilter;

	LPTSTR lpstrCustomFilter; DWORD nMaxCustFilter; DWORD nFilterIndex; LPTSTR lpstrFile; DWORD nMaxFile; LPTSTR lpstrFileTitle; DWORD nMaxFileTitle; LPCTSTR lpstrInitialDir; LPCTSTR lpstrTitle; DWORD Flags; WORD nFileOffset; WORD nFileExtension; LPCTSTR lpstrDefExt; DWORD lCustData; LPOFNHOOKPROC lpfnHook; LPCTSTR lpTemplateName; } OPENFILENAME; ويجب تغييرها حسب رغبتك لأنواع الملفات التي تريد قبل استدعاء الدالة.
);	انتهت.

1.9 WaitForDebugEvent

BOOL WaitForDebugEvent(	دالة انتظار حصول حدث من الأحداث الموجودة في البنية DEBUG_EVENT .
LPDEBUG_EVENT lpDebugEvent	مؤشر البنية DEBUG_EVENT وسنشرحه بالتفصيل.
DWORD dwMilliseconds	كم من الوقت تريد للدالة أن تنتظر؟ مثلاً: تخيل أنه لن يحدث أي حدث في برنامجنا، وتركنا هذه الدالة تنتظر!! أكيد سيبقى الـ Debbuger الخاص بنا مجمداً!! إن كنت تريد تفادي ذلك يمكنك أن تحدد وقتاً لإنهاء الدالة في حالتنا سنتركها دائما تنتظر، وسنجعل هذا البارومتر INFINITE .. انتظر دائماً حتى يحصل حدث..
);	انتهت.

1.9.1. البنية DEBUG_EVENT

هذه البنية تحتوي على جميع الأحداث الممكن أن تطرأ في برنامجنا المحمل الى الذاكرة، وبنيتها كالتالي:

typedef struct _DEBUG_EVENT {	بنية تحتوي جميع الأحداث الممكنة أثناء تنقيحالبرنامج المحمل في الذاكرة
DWORD dwDebugEventCode;	يحتوي كود الحدث الذي وقع أثناء تنقيح برنامج، ويمكن أن يكون القيم التالية: ⊙ EXCEPTION_DEBUG_EVENT : حدوث استثناء في البرنامج، وأشكاله كثيرة ضمن البنية u . سنتطرق لها بالتفصيل، وبالضبط في u.Exception . ⊙ CREATE_PROCESS_DEBUG_EVENT : البرنامج تم تحميله من أجل التنقيح . ترجع المعلومات في البنية u . بالضبط في u.CreateProcessInfo . ⊙ CREATE_THREAD_DEBUG_EVENT : البرنامج قيد التنقيح، إنشاء Thread جديدة. ترجع معلوماتها في البنية u. u.CreateThread . ⊙ EXIT_THREAD_DEBUG_EVENT : هناك Thread أغلقت في برنامج قيد التنقيح. وترجع معلوماتها في البنية u . بالضبط u.ExitThread . ⊙ EXIT_PROCESS_DEBUG_EVENT : البرنامج قيد التنقيح قد تم إغلاقه، وترجع معلومات طريقة الإغلاق في البنية u . بالضبط u.ExitProcess . ⊙ LOAD_DLL_DEBUG_EVENT : البرنامج قيد التنقيح قد حمل مكتبة الى الذاكرة ومعلوماتها ضمن البنية u . بالضبط u.LoadDll . ⊙ UNLOAD_DLL_DEBUG_EVENT : البرنامج قيد التنقيح قد حرر مكتبة، وتوجد معلوماتها في البنية u . بالضبط u.UnloadDll ⊙ OUTPUT_DEBUG_STRING_EVENT : البرنامج قيد التنقيح قد تبادل معلومات مع المنقح، ويمكن ذلك بالدالة : DebugOutputString . ترجع المعلومات في البنية u . بالضبط u.DebugString . ⊙ RIP_EVENT : تقرير حول أخطاء النظام أثناء التنقيح، ويتم إرجاع التقرير في البنية u . بالضبط في : u.RipInfo
DWORD dwProcessId;	الـ PID الخاص بالبرنامج قيد التنقيح تحصل عليه الدالة بشكل تلقائي
DWORD dwThreadId;	الـ ID للـ Thread الأساسي الخاص بالبرنامج قيد التنقيح تحصل عليه الدالة بشكل تلقائي.

union {	البنية أو التركيبة u . وهي بنيات داخل هذه البنية الأساسية : سنشرح كل بنية على حدا وبالتفصيل.
EXCEPTION_DEBUG_INFO Exception;	أنواع ومعلومات الاستثناءات الممكنة في البرنامج
CREATE_THREAD_DEBUG_INFO CreateThread;	معلومات الــ Threads التي ينشئها البرنامج قيد التنقيح
CREATE_PROCESS_DEBUG_INFO CreateProcessInfo;	معلومات البرنامج المحمل في الذاكرة بعد تحميله من أجل التنقيح
EXIT_THREAD_DEBUG_INFO ExitThread;	معلومات الــThreads التي أغلقها البرنامج قيد التنقيح.
EXIT_PROCESS_DEBUG_INFO ExitProcess;	معلومات شكل الخروج من برنامج قيد التنقيح
LOAD_DLL_DEBUG_INFO LoadDll;	معلومات حول المكتبة التي حملها برنامج قيد التنقيح
UNLOAD_DLL_DEBUG_INFO UnloadDll;	معلومات المكتبة التي حررها برنامج قيد التنقيح
OUTPUT_DEBUG_STRING_INFO DebugString;	معلومات حول دالة DebugOutPutString . (لن نتطرق لها)
RIP_INFO RipInfo;	معلومات حول أخطاء النظام أثناء التنقيح (لن نتطرق لها)
} u;	
} DEBUG_EVENT;	

1.9.2. البنية u

هذه البنية هي الكل في الكل للحصول على تفصيلات الأحداث التي تحصل أثناء تنقيحبرنامج، وسنتطرق لها بالتفصيل إن شاء الله .

EXCEPTION_DEBUG_INFO

بنية تحتوي تفصيل حول الاستثناء الذي حدث في البرنامج، وهي كالتالي :

typedef struct _EXCEPTION_DEBUG_INFO {	
EXCEPTION_RECORD ExceptionRecord;	بنية تحتوي أنواع الاستثناءات الممكنة أين وكيف وقعت؟.. بنيتها كالتالي : typedef struct _EXCEPTION_RECORD { DWORD ExceptionCode; DWORD ExceptionFlags; struct _EXCEPTION_RECORD *ExceptionRecord; PVOID ExceptionAddress; DWORD NumberParameters; DWORD ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS]; } EXCEPTION_RECORD;

حيث :

ExceptionCode :

نوع الاستثناء الذي حدث وسنشرح الشائعة منها فقط.

- EXCEPTION_ACCESS_VIOLATION

استثناء أثناء محاولة برنامج قيد التنقيح الوصول الى عنوان ذي حماية معينة؛ مثلاً نريد الكتابة في عنوان مخصص للقراءة فقط، أو تنفيذ عنوان خاص للكتابة فقط

- EXCEPTION_BREAKPOINT

حدوث استثناء أثناء تنفيذ عنوان يحتوي نقطة توقف من نوع مقاطعة، وهي في الغالب التعليمية Int3 . هذه الأخيرة تحدث استثناء من نوع نقطة توقف، وهو نفسه ما تعتمده OllyDbg لوضع نقطة توقف .

- EXCEPTION_FLT_DIVIDE_BY_ZERO

استثناء أثناء محاولة برنامج قيد التنقيح إنجاز عملية قسمة على 0 .

- EXCEPTION_SINGLE_STEP

استثناء يحدث أثناء التتبع؛ مثل خاصية Tracing في OllyDbg تعتمد هذا الاستثناء، ويمكننا من معرفة عدد التعليمات التي نفذت في البرنامج من فتحه حتى إغلاقه أو كما تريد

- EXCEPTION_STACK_OVERFLOW

واضحة من اسمها، استثناء أثناء حصول Stack OverFlow .

للمزيد من المعلومات حول جميع الاستثناءات ابحث في MSDN وهي كالتالي :

EXCEPTION_ACCESS_VIOLATION
EXCEPTION_ARRAY_BOUNDS_EXCEEDED
EXCEPTION_BREAKPOINT
EXCEPTION_DATATYPE_MISALIGNMENT
EXCEPTION_FLT_DENORMAL_OPERAND
EXCEPTION_FLT_DIVIDE_BY_ZERO
EXCEPTION_FLT_INEXACT_RESULT
EXCEPTION_FLT_INVALID_OPERATION
EXCEPTION_FLT_OVERFLOW
EXCEPTION_FLT_STACK_CHECK
EXCEPTION_FLT_UNDERFLOW
EXCEPTION_ILLEGAL_INSTRUCTION
EXCEPTION_IN_PAGE_ERROR
EXCEPTION_INT_DIVIDE_BY_ZERO
EXCEPTION_INT_OVERFLOW
EXCEPTION_INVALID_DISPOSITION
EXCEPTION_NONCONTINUABLE_EXCEPTION
EXCEPTION_PRIV_INSTRUCTION
EXCEPTION_SINGLE_STEP
EXCEPTION_STACK_OVERFLOW

لا يهمنا الباقي، يهمنا فقط هذا البارومتر: **ExceptionAddress** : العنوان الذي حدث عنده استثناء	
لن نحتاجه.	DWORD dwFirstChance;
	} EXCEPTION_DEBUG_INFO;

CREATE_THREAD_DEBUG_INFO

تحتوي معلومات الــ Thread التي أنشأها برنامج قيد التنقيح، وهي كالتالي :

	typedef struct _CREATE_THREAD_DEBUG_INFO {
مقبض الــ Thread التي أنشأها البرنامج قيد التنقيح	HANDLE hThread;
لن نحتاجه	LPVOID lpThreadLocalBase;
العنوان الذي تنطلق منه الــThread الجديدة في برنامج قيد التنقيح	LPTHREAD_START_ROUTINE lpStartAddress;
	} CREATE_THREAD_DEBUG_INFO;

CREATE_PROCESS_DEBUG_INFO

بنية تحتوي على معلومات الــ**Process** أثناء إنشاءه من أجل التنقيح	typedef struct _CREATE_PROCESS_DEBUG_INFO {
مقبض الملف على مستوى القرص الصلب	HANDLE hFile;
مقبض الــ Process	HANDLE hProcess;
مقبض الــ Thread الأساسي في الــProcess	HANDLE hThread;
ImageBase واضحة. (راجع دروس الــPE Format).	LPVOID lpBaseOfImage;
لن نحتاجه.	DWORD dwDebugInfoFileOffset;
لن نحتاجه.	DWORD nDebugInfoSize;
لن نحتاجه.	LPVOID lpThreadLocalBase;
نقطة الدخول لبرنامج قيد التنقيح EP= ImageBase + OEP .	LPTHREAD_START_ROUTINE lpStartAddress;
مسار الملف قيد التنقيحASCII .	LPVOID lpImageName;
مسار الملف Unicode .	WORD fUnicode;
	} CREATE_PROCESS_DEBUG_INFO;

EXIT_THREAD_DEBUG_EVEN

typedef struct _EXIT_THREAD_DEBUG_INFO {	حالة إغلاق الـ Thread
DWORD dwExitCode;	في حالة نجاح الإغلاق بشكل سليم تكون هذه القيمة تساوي0.
} EXIT_THREAD_DEBUG_INFO;	

EXIT_PROCESS_DEBUG_EVEN

typedef struct _EXIT_PROCESS_DEBUG_INFO {	تعطي طريقة الخروج من التنقيح
DWORD dwExitCode;	كود حالة الخروج من التنقيح
} EXIT_PROCESS_DEBUG_INFO;	

LOAD_DLL_DEBUG_INFO

typedef struct _LOAD_DLL_DEBUG_INFO {	بنية تحتوي معلومات المكتبة التي حملها البرنامج أثناء التنقيح.
HANDLE hFile;	مقبضها.
LPVOID lpBaseOfDll;	الـ ImageBase للمكتبة.
DWORD dwDebugInfoFileOffset;	لن نحتاجه.
DWORD nDebugInfoSize;	لا نحتاجه.
LPVOID lpImageName;	مسار المكتبة واسمها.
WORD fUnicode;	مسار المكتبة واسمها Unicode .
} LOAD_DLL_DEBUG_INFO;	

UNLOAD_DLL_DEBUG_EVEN

typedef struct _UNLOAD_DLL_DEBUG_INFO {	بنية تحتوي على معلومات المكتبة المحررة أثناء تنقيحالبرنامج.
LPVOID lpBaseOfDll;	عنوانها في الذاكرة ImageBase.
} UNLOAD_DLL_DEBUG_INFO;	

ContinueDebugEvent 1.10

هذه الدالة لإتمام تنقيح البرنامج وكأننا نعمل Trace للبرنامج أي تتبعه...

BOOL ContinueDebugEvent (	دالة ترجع TRUE إذا نجحت والعكس بالعكس
DWORD dwProcessId	الـ Id الخاص ببرنامج قيد التنقيح . نحصل عليه من الدالة CreateProcess أتذكرها؟ ملاحظة مهمه : هو Id ولـ hProcess يجب التمييز بينهم

DWORD dwThreadId	الـ Id الخاص بـ Thread الأساسية. ملاحظة : هو Id وليس hThread .
DWORD dwContinueStatus	كيف ستكمل الدالة التنقيح وهي بهذه الأشكال : DBG_CONTINUE توقف البرنامج في المنقح إذا حدث أي استثناء DBG_EXCEPTION_NOT_HANDLED أكمل تنفيذ البرنامج بدون التوقف عند الاستثناءات ملاحظة : إذا استعملت هذه الخاصية وحدث استثناء، فسوف يغلق البرنامج بالخطأ، لذلك يجب أن تحذر في التعامل مع هذه الخصائص
);	

DebugActiveProcess 1.11

كما نعلم أنه في OllyDbg هنالك خاصية عمل Attach لبرنامج، وهي تعني تنقيح برنامج محمل أصلاً للذاكرة. لعمل ذلك نحتاج لهذه الدالة DebugActiveProcess .

BOOL DebugActiveProcess (	دالة عمل **Attach** لبرنامج من أجل التنقيح. ترجع **TRUE** إذا نجحت. والعكس بالعكس.
DWORD dwProcessId	الـ Pid الخاص بالبرنامج الذي نريد تنقيحه، ويمكنك الحصول عليه بأحد محررات الـ process مثل TaskMan أو Kernel Detective
);	

بهذا نكون قد أنهينا فقرة شرح الدوال، وسنمر للفقرة التالية المخصصة للأمثلة

2. ثانياً : حلقة التنقيح

2.1 طريقة عمل Debug لبرنامج

في هذه الفقرة سنكتب خوارزمية وكوداً بسيطاً لـ Debugger بسيط، وهو سيتكرر في كل الـ Debbugers التي سنطرح، والتي توجد الآن.

2.1.1. نافذة اختيار البرامج من أجل التنقيح

كما شرحنا من قبل فالدالة المسؤولة عن نافذة اختيار البرامج هي GetOpenFileNameA ويمكن استغلالها بالشكل التالي :

أولاً تعريف متغير من نوع البنية التي تحتاج هذه الدالة بالشكل التالي:

```
OPENFILENAME ofn;
```

ثم تغيير خصائص البنية حسب شكل الملفات التي نريد تنقيحها، وهي غالباً من هيئة exe . ثم ستحتاج هذه المتغير الذي سنحفظ فيه مسار البرنامج المختار.

أثناء تعريف البنية في بعض المجمعات وفي الغالب لن يكون مكانها فارغاً، وهذا سيؤثر على الدالة، لذلك وجب تصفير البنية بالدالة RtlZeroMemory وكذلك الأمر لكل بنية عرفناها أو متغير كبير.

كما قلنا فالبنية تحتاج لمتغير تحفظ فيه المسار وسنعرفه كالتالي:

```
char FileTarget[256];
```

ثم نصفر البنية والمتغير :

```
ZeroMemory (&ofn,sizeof(ofn));
ZeroMemory (FileTarget,sizeof(FileTarget));

ofn.lStructSize = sizeof(ofn);
ofn.lpstrFilter = "Win32\0*.exe";
ofn.lpstrFile = FileTarget;
ofn.nMaxFile = 256;
ofn.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST;
```

ثم نستدعي الدالة كالتالي مع اختبارها :

```
if(GetOpenFileNameA(&ofn))
{
      //   الكود الذي نريد تنفيذه إن نجحت الدالة
}
```

2.1.2. تحميل برنامج للذاكرة من أجل التنقيح

وماذا بعد إن قمنا باختيار البرنامج الذي نريد تنقيحه؟

يجب تحميله الى الذاكرة بخاصية التنقيح، وذلك ما تفعله الدالة CreateProcess . سنستغلها كالتالي :

لكن قبل ذلك أتذكر الدالة GetStartupInfo؟ سيكون الناتج هكذا :

أولاً يجب تعريف البنيات التي تحتاج الدالتين ثم تصفيرهما كالتالي:

```
STARTUPINFO sinfo;
PROCESS_INFORMATION pinfo;
BOOL Result=FALSE ;

ZeroMemory(&pinfo,sizeof(pinfo));
ZeroMemory(&sinfo,sizeof(sinfo));
```

لماذا قمت بتعرف المتغير Result ؟ تذكر أن الدالة CreateProcess ترجع قيمة BOOL .

```
GetStartupInfo(&sinfo);
Result=CreateProcess(FileTarget,0,0,0,0,DEBUG_PROCESS+
DEBUG_ONLY_THIS_PROCESS,0,0,&sinfo,&pinfo);
```

ها قد أصبح برنامجنا متوقف في الذاكرة. ما الخطوة التالية ؟ تابع ...

2.1.3. الدخول لحلقة التنقيح

هذا هو الجزء المهم في المنقح لذلك ركز جيداً في هذه الفقرة

الحلقة بالشكل التالي :

```
while(Result)
{

}
```

أي أنه إذا نجح في تحميل البرنامج للذاكرة فسيدخل الحلقة. ولكن متى يخرج من الحلقة ؟ سنعرف الجواب قريباً

ماذا بعد أن دخل الحلقة؟ كيف هي حالة البرنامج ؟ أكيد هو متوقف !! لكن توقف من الدالة CreateProcess لذلك وجب إدخاله لحلقة التنقيح وذلك بالدالة WaitForDebugEvent نظن أنك تتذكرها جيداً؟

هذه الدالة تحتاج بارامتري تم شرحهما، ويجب تعريف البنية المستخدمة في هذه الدالة كالتالي:

```
DEBUG_EVENT DebugEvent;
```

ثم نستدعي الدالة داخل الحلقة :

```
while(Result)
{
       WaitForDebugEvent(&DebugEvent,INFINITE);
}
```

الآن ماذا بعد أن أصبح في حلقة التنقيح؟

يجب مراقبة جميع الأحداث التي تحدث في البرنامج ، أتعرف كيف يتم ذلك ؟ راجع القيم الممكنة في المتغير dwDebugEventCode

ولذلك سنقوم بعمل Switch Case لهذا المتغير حسب القيم الممكنة له ونراقب الأحداث. ثم إكمال التنقيح في حالة لم يحدث أي حدث من الأحداث الممكنة، وذلك بالدالة : ContinueDebugEvent راجعها جيداً.

يمكن استعمالها كالتالي لتجاوز الاستثناءات :

```
ContinueDebugEvent(DebugEvent.dwProcessId,
DebugEvent.dwThreadId,DBG_EXCEPTION_NOT_HANDLED);
```

وبالتالي تكون الحلقة كاملة بالشكل التالي :

بعد أن نضيف له تجاوز الاستثناءات كلها، وذلك بالدالة ContinueDebugEvent بالخاصية DBG_CONTINUE .

```
while(Result)
     {
      WaitForDebugEvent(&DebugEvent,INFINITE);
          switch(DebugEvent.dwDebugEventCode)
          {
               case CREATE_PROCESS_DEBUG_EVENT:

               break;

               case EXCEPTION_DEBUG_EVENT:
                    ContinueDebugEvent( DebugEvent.dwProcessId,
                                        DebugEvent.dwThreadId,
                                        DBG_CONTINUE);

               break;

               case CREATE_THREAD_DEBUG_EVENT:

               break;
```

```
            case EXIT_PROCESS_DEBUG_EVENT:
                  Result = FALSE ;
            break;
        }

    Result=ContinueDebugEvent( DebugEvent.dwProcessId,
                               DebugEvent.dwThreadId,
                               DBG_EXCEPTION_NOT_HANDLED);
}

CloseHandle(pinfo.hProcess);
CloseHandle(pinfo.hThread);
ExitProcess(0);
```

لا ننسى إغلاق المنقح وإغلاق المقابض بعد الخروج من الحلقة، وأن شرط الخروج من الحلقة هو إغلاق البرنامج الذي ننقح

وبهذا نكون قد أنهينا حلقة التنقيح. وسنمر لفقرة الأمثلة :

الكود بشكل كامل ستجده في المرفقات باسم Debug1.cpp	

3. ثالثاً : كيفية وضع نقطة توقف BP من نوع استثناء في التعليمية INT3 ؟

هذه هي الفقرة الأهم في الدرس، وسنقوم بتقسيمها إلى قسمين أساسيين وهما:

1) وضع نقطة التوقف وإزالتها في البرنامج في حلقة التنقيح

2) تتبع البرنامج للوصول الى نقطة التوقف وجلب معلومات جميع المسجلات.

3.1. كيفية وضع نقطة التوقف وإزالتها في البرنامج في حلقة التنقيح

والآن وبعد أن عرفنا كيفية جعل برنامج ما في حلقة التنقيح، سنمر للفقرة الأكثر تشويقاً، وهي طريقة وضع نقطة توقف كما تفعل في OllyDbg تماما لوضع نقطة توقف بـــ(F2) .

كما نعلم فإن التعليمية int3 تحدث استثناءً في المعالج، وبما أن حلقة التنقيح تتيح لنا التوقف عند الاستثناءات، فببساطة لوضع نقطة توقف عند عنوان معين يكفي أن نجعل ذلك العنوان ينفذ التعليمية Int3 .

3.1.1. كيفية وضع نقطة توقف بالتعليمية INT3 في عنوان معين

كما نعلم أنه عند كل عنوان تنفذ تعليمية معينة أو بشكل أدق ينفذ Op Code معين، الـ Op Code للتعليمية int3 هو 0xCC (0CCh) .

لذلك سنستعمل الدالة WriteProcessMemory لكتابة التعليمية 0xCC عند العنوان المراد التوقف عنده، لكن قبل وضع نقطة توقف يجب أن نحفظ التعليمية الأصلية لكي نرجع التنفيذ للبرنامج، وسنحتاج لهذا الغرض للدالة: ReadProcessMemory .

```
void Set_int3_BP(HANDLE hProcess,void* Address,char *OriginalOP)
{
    char int3_OP=0xCC;
    ReadProcessMemory(hProcess,Address,OriginalOP,1,0);
    WriteProcessMemory(hProcess,Address,&int3_OP,1,0);
}
```

كما ترى فقد قمنا بتعريف دالة بثلاثة باروميترات هم على التوالي :

1) hProcess : مقبض البرنامج المحمل للذاكرة.
2) Address : العنوان الذي نريد وضع نقطة توقف عنده.
3) OriginalOP : لنحفظ فيه التعليمية الأصلية قبل وضع نقطة توقف

3.1.2. كيفية إزالة نقطة التوقف وإرجاع التحكم للبرنامج

كيفية إزالة نقطة التوقف؟

نفترض أننا نريد وضع نقطة توقف في العنوان 0x004010E4 فأثناء تنفيذ التعليمية Int3 عند هذا العنوان يحصل استثناء. لاحظ أننا قلنا عند تنفيذها، أي أن هذه التعليمية نفذت وأن المؤشر نحو مقطع الكود EIP قد زاد بواحد لأن الـ OP Code للـ int3 ما هو إلا بايت واحد.

إذن الاستثناء حصل الآن و EIP = 0x004010E5 والمنقح الذي كتبنا أوقف البرنامج عند الحدث EXCEPTION_DEBUG_EVENT. الآن كيف نعرف أن الاستثناء حصل في هذا العنوان بالضبط؟ وكيف نزيل نقطة التوقف؟ لفعل ذلك ما علينا سوى اتباع الطريقة التالية :

- أولاً يجب أن نختبر الاستثناء لنعرف أنه من نوع EXCEPTION_BREAKPOINT . إذا كان كذلك نقوم بجلب الـ Context للبرنامج قيد التنقيح بالدالة GetThreadContext بعد أن نجعل ContextFlags=CONTEXT_FULL لجلب البنية بأكملها .

ثم بعد ذلك نقارن الـــ EIP من الـــ Context إذا كان هو نفسه العنوان الذي نريد التوقف عنده . في حالتنا هو 0x004010E5 تذكر أنه قد أضفنا 1 للعنوان لأن التعليمية نفذت.

هاهو الآن البرنامج متوقف تماماً عند العنوان الذي نريد، الآن سنقوم بإزالة نقطة التوقف، وذلك بكتابة دالة بسيطة تقوم بإزالتها، وهي واضحة :

```
void Remove_int3_BP(CONTEXT *cntxt,HANDLE hThread,HANDLE hProcess,void*
Address,char OriginalOP)
{
    WriteProcessMemory(hProcess,Address,&OriginalOP,1,0);
    cntxt->ContextFlags=CONTEXT_FULL;
    GetThreadContext(hThread,cntxt);
    cntxt->Eip--;
    SetThreadContext(hThread,cntxt);
}
```

سنوضح كيف تعمل هذه الدالة؟ في الأول تقوم بإرجاع التعليمية الأصلية التي غيرناها بـ int3 ثم تقوم بجلب الـــ Context الحالي للبرنامج . تذكر أننا نقف عند عنوان يزيد بـــ 1 عن العنوان الذي توقفنا عنده، لذلك وجب أن ننقص الـــ EIP بواحد لكي نعود للتعليمة الأصلية، ثم نرجع التحكم للبرنامج بالدالة SetThreadContext .

والسؤال الذي يطرح نفسه : متى يجب أن نضع نقطة توقف؟

يمكنك وضعها مباشرة بعد تحميل البرنامج الهدف للذاكرة، هكذا:

```
Result=CreateProcess(  FileTarget,
                       0,0,0,0,
                       DEBUG_PROCESS+ DEBUG_ONLY_THIS_PROCESS,
                       0,0,
                       &sinfo,
                       &pinfo);
Set_int3_BP(pinfo.hProcess,(void *)0x004010E4,&OriginalOP);
```

أو (تذكر قلنا: "أو") يمكنك وضعها في الحدث CREATE_PROCESS_DEBUG_EVENT هكذا :

```
switch(DebugEvent.dwDebugEventCode)
    {
    case CREATE_PROCESS_DEBUG_EVENT:
                    Set_int3_BP(pinfo.hProcess,(void *)0x004010E4,&OriginalOP);
          break;
```

متى يجب إزالة نقطة التوقف؟

يمكن إزالتها أثناء التوقف عند الاستثناء، كالتالي :

```
case EXCEPTION_DEBUG_EVENT:
   if(DebugEvent.u.Exception.ExceptionRecord.ExceptionCode==EXCEPTION_BREAKPOINT)
   {
        cntxt.ContextFlags=CONTEXT_FULL;
        GetThreadContext(pinfo.hThread,&cntxt);
        if(cntxt.Eip==0x004010E5)
           {
                MessageBox(0,"We Are on the BP","On BP",MB_OK);
                Remove_int3_BP(   &cntxt,
                                  pinfo.hThread,
                                  pinfo.hProcess,
                                  (void *)0x004010E4,
                                  OriginalOP);
           }
   }
   ContinueDebugEvent(DebugEvent.dwProcessId,
                      DebugEvent.dwThreadId,
                      DBG_CONTINUE);
break;
```

3.2. ثالثا: مثال تطبيقي متقدم



تجد الكود كاملاً في المرفقات لمثال يقتنص السريال من ضحية مرفق مع السورس وهو واضح جداً.

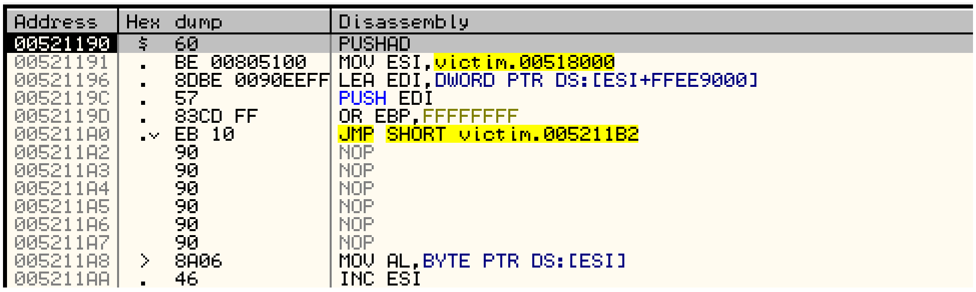
وأيضاً سنجد مثالاً يعطي قيمة جميع المسجلات أثناء التوقف في عنوان ما.

الآن سنقوم بدراسة مثال متقدم قليلاً، نوضح من خلاله طريقة تتبع ووضع نقط توقف على برنامج مضغوط وقنص السريال أيضاً، وسنقوم بتطبيق كل ما تعلمناه في هذا الدرس على هذا المثال؛ حيث سنراقب أغلب الأحداث به.

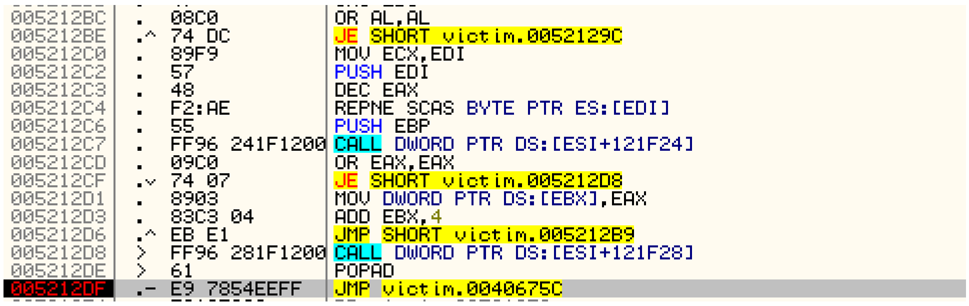
افتح الضحية وستراها كالتالي :



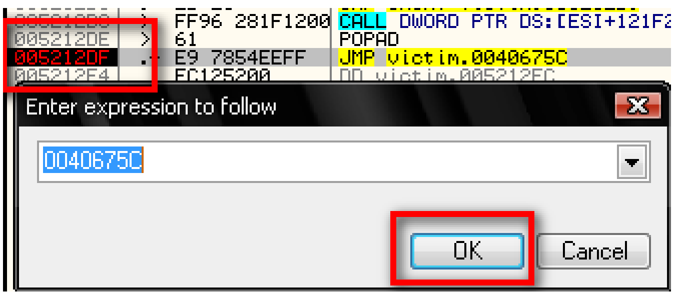
يطلب الاسم والسريال وهنالك رسالة خطأ. الآن حمله في OllyDbg :



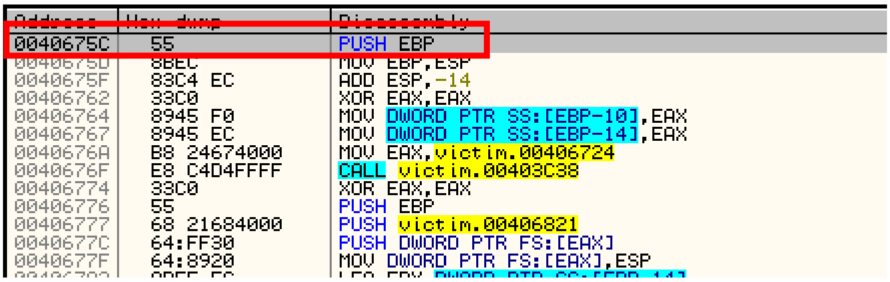
أنت تعرف كيف تفك ضغط UPX يدوياً صحيح؟! انزل الى القفزة المميزة لهذا الضاغط ثم اضغط F9 :



القفزة عند العنوان 0x05212DF0 تذكره جيداً. و الآن لا تضغط F8 بل اتجه إلى العنوان الذي تذهب له القفزة :



ستنتقل الى العنوان 0x040675C0:



جميل!! الآن أصبح الكود واضحاً، إذن ماذا حصل حين وضعنا نقطة التوقف؟!

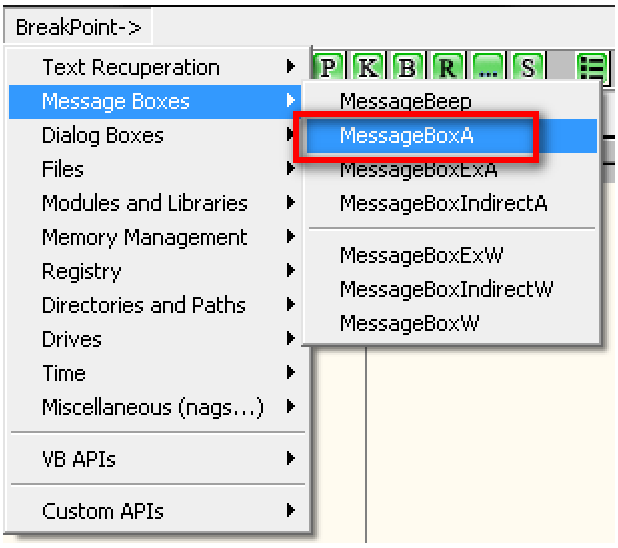
كما تعلم فـــ UPX برنامج ضغط. ما يحصل قبل تنفيذ القفزة هو فك ضغط كود البرنامج في الذاكرة ثم تنفيذه بعدها.

الآن لنضغط F9 ولنشاهد أين يختبئ السريال؟

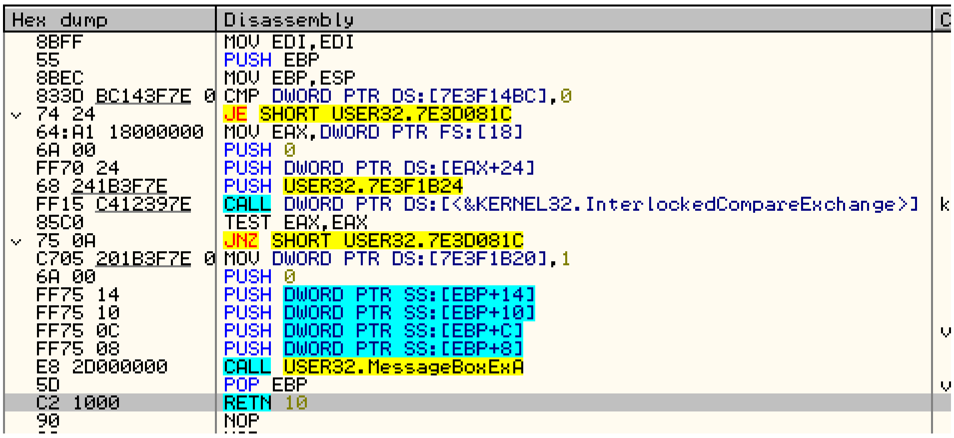
بعد الضغط على F9 ضع نقطة توقف عند MessageBox من خلال CommandLine في OllyDbg هكذا :



أو من خلال إضافة أخينا Zool@nder هكذا :



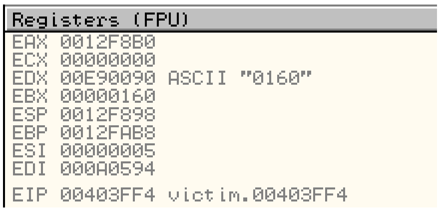
ثم اكتب أي اسم وأي سريال، وليكن AT4RE و السريال هو 1111111111111 ثم اضغط "تحقق من التسجيل" وسيتوقف OllyDbg في هذا المكان :



أزل نقطة التوقف من MOV EDI,EDI ثم ضعها في RETN 10 واضغط F9 ستجد أن رسالة الخطأ قد ظهرت لك، موافق وبعدها سيتوقف OllyDbg عند RETN 10 .

اضغط F8 وستجد نفسك عند مصدر رسالة الخطأ.

الآن ضع نقطة توقف عند هذا العنوان 00403FF4 ثم أعد إدخال الاسم والتسجيل، وعندها سنشاهد التالي (انظر المسجلات) :



هاهو السريال الخاص بالاسم AT4RE إنه 0160 جربه.. إنه يعمل !! الآن يجب أن نتذكر ما يلي :

1) عنوان القفزة لفك الكود في الذاكرة: 0x05212DF .
2) العنوان الذي نريد التوقف عنده لقنص السريال، وهو: 0x0403FF4 .
3) السريال يوجد في العنوان الذي في المسجلEDX .

أغلق OllyDbg ودعنا نفعل ذلك برمجياً.

أولاً : سنحمل البرنامج من أجل التنقيح ونظنك تعرف كيف تقوم بذلك؟!.

ثانياً : سنقوم بوضع نقطة توقف عند عنوان القفزة 0x05212DF وحين يتوقف البرنامج نزيلها ثم نضع نقطة أخرى عند العنوان 0x0403FF4 وحين يتوقف عند هذه الأخيرة نقوم بقنص المسجلات ومعلومات التسجيل :

```
case CREATE_PROCESS_DEBUG_EVENT:
    Set_int3_BP(pinfo.hProcess,(void *)0x005212DF,&OriginalOP);
break;

case EXCEPTION_DEBUG_EVENT:
    if(DebugEvent.u.Exception.ExceptionRecord.ExceptionCode==EXCEPTION_BREAKPOINT)
    {
        cntxt.ContextFlags=CONTEXT_FULL;
        GetThreadContext(pinfo.hThread,&cntxt);
        if(cntxt.Eip == 0x005212DF + 1)
        {
             Remove_int3_BP(&cntxt,
                                  pinfo.hThread,
                                  pinfo.hProcess,
                                  (void *)0x005212DF,
                                  OriginalOP);

             Set_int3_BP(pinfo.hProcess,(void *)0x00403FF4,&OriginalOP);
        }
        else if(cntxt.Eip == 0x00403FF4 + 1)
        {
             wsprintf(RegInformation,"EAX = %.8X\n"
                                       "ECX = %.8X\n"
                                       "EDX = %.8X\n"
                                       "EBX = %.8X\n"
                                       "ESP = %.8X\n"
                                       "EBP = %.8X\n"
                                       "ESI = %.8X\n"
                                       "EDI = %.8X",
                                  cntxt.Eax,
                                  cntxt.Ecx,
                                  cntxt.Edx,
                                  cntxt.Ebx,
                                  cntxt.Esp,
                                  cntxt.Ebp,
                                  cntxt.Esi,
                                  cntxt.Edi);

             MessageBox(0,RegInformation,"Registers",MB_OK | MB_ICONINFORMATION);
             ReadProcessMemory(pinfo.hProcess,(void *)cntxt.Edx,Serial,5,0);
             MessageBox(0,Serial,"Your Serial Is :",0);
             Remove_int3_BP ( &cntxt,
                                  pinfo.hThread,
                                  pinfo.hProcess,
                                  (void *)0x00403FF4,
                                  OriginalOP );
        }
    }
    ContinueDebugEvent(DebugEvent.dwProcessId,
                           DebugEvent.dwThreadId,
                           DBG_CONTINUE);
break;
```

الآن سنضيف بعض الأحداث؛ مثلاً إذا حمل البرنامج دالة ما، وسنقوم بجلب الـــ ImageBase لها، وأيضاً سنظهر رسالة تعطي كود حالة خروج البرنامج من التنقيح :

```
case LOAD_DLL_DEBUG_EVENT:
```

```
    wsprintf(SomeInfo,"The Program Loads a new Library its ImageBase = 
%.8X",DebugEvent.u.LoadDll.lpBaseOfDll);
    MessageBox(0,SomeInfo,"Dll ImageBase",MB_OK);
break;
```

جلب بعض معلومات PE أثناء تحميل الملف للذاكرة :

```
case CREATE_PROCESS_DEBUG_EVENT:
    Set_int3_BP(pinfo.hProcess,(void *)0x005212DF,&OriginalOP);
    wsprintf(SomeInfo,"Program Loaded :\nImageBase = %.8X\nEP = %.8X",
             DebugEvent.u.CreateProcessInfo.lpBaseOfImage,
             DebugEvent.u.CreateProcessInfo.lpStartAddress);
    MessageBox(0,SomeInfo,"Some Pe Info",MB_OK);
break;
```

طبع رسالة أثناء الخروج من التنقيح، وأيضاً كود حالة الخروج من التنقيح:

```
case EXIT_PROCESS_DEBUG_EVENT:
      wsprintf(SomeInfo,"Program Closed , its state = %.8X",
               DebugEvent.u.ExitProcess.dwExitCode);
    MessageBox(0,SomeInfo,"State",MB_OK);
    Result=FALSE;
break;
```

يمكنك التمرن على باقي الأحداث إن أردت السورس كود في المرفقات.

ماذا الآن لو أردنا عمل Attach لبرنامج من أجل التنقيح ؟

الأمر بسيط ويمكن القيام به بحذف الأمرين CreateProcess و GetOpenFileNameA وتعويضهما بالدالة DebugActiveProcess التي يمكن استغلالها بهذا الشكل؛ مثلاً افتح برنامج Kernel Detective لتحصل على PID برنامج لأجل تنقيحه :

C:\Program Files\Notepad++\notepad++.exe	1576	704	0x8181A020
C:\Program Files\K-Lite Codec Pack\Media Player Classi...	1792	1392	0x816E3020
C:\Program Files\Microsoft Office\OFFICE11\WINWOR...	1928	1392	0x816F4688
C:\Program Files\CodeBlocks\codeblocks.exe	2004	1392	0x8164C020

ثم نعطه للدالة DebugActiveProcess كالتالي :

```
DebugActiveProcess((DWORD)1576) ;
```

ثم ندخله لحلقة التنقيح.

إلى هنا ننهي الدرس والصلاة والسلام على أشرف الخلق سيدنا محمد عليه الصلاة والسلام

السكريبتينغ في OllyDbg

Scripting In OLLYDBG : OdbgScript

4. المقدمة

OdbgScript : هي لغة برمجة خاصة ببرنامج التنقيح OllyDbg وهي شبيهة إلى حدٍ ما بلغة التجميع.

ككل لغة سكريبت OdbgScript له أوامر ومعرفات خاصة به تجعل عملية التنقيح أمراً سهلاً خصوصاً عندما يتطلب الأمر تنفيذ عمليات آلية أو متكررة أو الاثنين معاً.

سنتطرق في هذا الدرس إلى التعريف بكل الأوامر والمتغيرات التابعة لـ OdbgScript ، كلٍ على حدة، مع التوقف عند بعض الأمثلة السريعة والعملية، والتي يمكن حقن معظمها مباشرة في ملف سكريبت نصي واستدعاؤها للتجريب.

5. أساسيات يجب معرفتها

هي مشابهة لنظيرتها في لغة التجميع مع اختلافات بسيطة :

- يتعامل OdbgScript مع كل القيم الثابتة على أنها قيم ستة عشرية.
- عند استعمال قيم عشرية ينبغي أن تتبع بنقطة.
- عند استعمال عبارات ستة عشرية، يجب حصرها بين الرمز # (مثال: #4A00#).
- يمكن استعمال الرمز ? لتعويض القيم المجهولة (مثال: #4?00#).
- كباقي لغات البرمجة، OdbgScript يستعمل المعاملات التالية: + – * / & | ^ < > حيث :

& يقابل AND
| يقابل OR
^ يقابل XOR
> يقابل SHR
< يقابل SHL

المتغيرات المحجوزة: هي أسماء خاصة يمنع أن تمنح لأي متغير آخر:

- $RESULT تحتوي دائماً القيمة المرجعة من الأوامر.

- $RESULT_1 القيمة المرجعة من بعض الأوامر.
- $RESULT_2 القيمة المرجعة من بعض الأوامر.
- $VERSION ترجع رقم إصدار OdbgScript.

6. التعليقات

عند إدخال تعليقات نصية على أي سكريبت يجب مراعاة ما يلي:

- التعليق يسبق بأحد الرمزين ; أو // .
- سطر التعليق المسبوق بالرمز ; يظهر في نافذة تنفيذ السكريبت.
- يمكن الاستغناء عن الرمزين السابقين شرط حصر نص التعليق بين */ و/* .

7. الجذاذات

- الجذاذة (label) : تسمية سطر معين من السكريبت لغرض القفز إليه عند تنفيذ السكريبت.
- لتعريف جذاذة يكفي كتابة الاسم المراد لها متبوع بالرمز نقطتان : .
- لاستدعائها يكتب اسمها بعد التعليمة المناسبة دون أن تتبع بالرمز نقطتان : .

8. أوامر OdbgScript

8.1. الصيغة العامة

الصيغة العامة لأي أمر ستكون على الشكل التالي :

```
INSTRUCTION param1, param2 [param3]
```

حيث :

INSTRUCTION	➢	التعليمة أو الأمر.
param1	➢	المعامل الأول.
param2	➢	المعامل الثاني.
param3	➢	المعامل الثالث، وهو اختياري وغير إلزامي؛ لوجوده بين معقوفتين.

8.2. التعليمات

:ADD		
	الصيغة	ADD dest, src

الأمر	البند	المحتوى	التعليق
	المفعول	إضافة src إلى dest وحفظ الناتج في dest	
	مثـــال	add x, "times"	// بعد العملية x= "100 times"
		mov x, 100	// x = 100
		add ecx, x	// إضافة 100 إلى ecx
		add [401000], 200	// إضافة 200 إلى محتوى العنوان 401000
AL:	الصيغة	AL	
	المفعول	تنفذ الأمر "Animate into"	
ALLOC:	الصيغة	ALLOC size	
	المفعول	تخصص مساحة قدرها size في الذاكرة. وترجع العنوان المخصص في $RESULT	
AN:	الصيغة	AN add	
	المفعول	تنفذ الأمر Analyze للعنوان addr	
	مثـــال	an eip	// = CTRL-A
AO:	الصيغة	AO	
	المفعول	تنفذ الأمر "Animate Over"	
ASK:	الصيغة	ASK question	
	المفعول	▪ تظهر نافذة نصية تحتوي العبارة question لجلب النص المدخل من طرف المستعمل. ▪ ترجع $RESULT = 0 عند الضغط على زر الإلغاء. ▪ ترجع كذلك طول النص في المتغير $RESULT_1.	
	مثـــال	mov eip, $RESULT "ask "Enter new EIP cmp $RESULT, 0 je **cancel_pressed** mov eip, $RESULT **cancel_pressed**: ret	//
ASM:	الصيغة	ASM addr, command	
	المفعول	تحقن الأمر command في العنوان addr	
	مثـــال	asm 4012FE, "xor eax,eax"	
ASMTXT:	الصيغة	ASMTXT addr, file	
	المفعول	تحقن كود ملف التجميع في العنوان addr	
	مثـــال	asm 4012FE, "mycode.asm"	
ATOI:	الصيغة	ATOI str [, base=16.]	

الأمر	الحقل	الكود	التعليق
	المفعول	تحويل العبارة الرقمية str من الأساس base (افتراضياً ستة عشري) إلى مقابلها الستة عشري. الناتج في $RESULT	
	مثـــال	atoi "30", 10.	// RESULT = 1E
		atoi "1110", 2	// RESULT = 0E
:BC			
	الصيغة	BC [addr]	
	المفعول	حذف نقطة توقف غير مشروطة عند العنوان addr	
	مثـــال	bc 6A0000	//
		bc eip	//
:BD			
	الصيغة	BD [addr]	
	المفعول	تعطيل نقطة توقف غير مشروطة عند العنوان addr	
	مثـــال	bd 6A0000	
		bd eip	
:BEGINSEARCH			
	الصيغة	BEGINSEARCH [start]	
	المفعول	نسخ الكود المنقح ابتداءً من العنوان start إلى الذاكرة. وتستعمل بالموازاة مع ENDSEARCH غالباً لتسريع عملية البحث بالأمر FIND.	
	مثـــال	mov count, 0	;
		mov start, eip	;
		beginsearch start	; نسخ ابتداءً من eip
		loop:	;
		find start, #60#	; البحث ابتداءً من eip عن التعليمة pushad
		cmp $RESULT,0	; فحص نتيجة البحث
		je end	; خروج إذا لم توجد
		mov start, $RESULT+1	; نقل العنوان التالي الذي وجدت فيه التعليمة إلى start
		add count, 1	; زيادة العداد
		jmp loop end :	;
		end :	;
		endsearch	; تفريغ الذاكرة
		msg count	; إظهار عدد مرات وجود التعليمة
:BP			
	الصيغة	BP addr	
	المفعول	تفعيل نقطة توقف غير مشروطة عند العنوان addr	
	مثـــال	bp 6A0000	
		bp eip	
:BPCND			
	الصيغة	BPCND addr, cond	

الأمر		
	المفعول	تفعيل نقطة توقف مشروطة بالشرط cond عند العنوان addr
	مثـــال	bpcnd 401000, "AL==1" توقف إذا المسجل AL=1
BPGOTO:		
	الصيغة	BPGOTO addr, label
	المفعول	القفز إلى الجُذَاذَة label بعد الوقوف على نقطة التوقف عند العنوان addr وهي مشابهة للأمر EOB
	مثـــال	mov addr, 64A000 bp addr bpgoto addr, **notice** ; عند حدوث int3 exception, ... ; التعليمات تكمل انطلاقاً من notice ... **notice**: msg "Break Point set correctly"
BPHWC		
	الصيغة	BPHWC [addr]
	المفعول	حذف نقطة توقف هاردوير عند العنوان addr
	مثـــال	bphwc 64A000
BPHWS		
	الصيغة	BPHWS addr, [mode]
	المفعول	وضع نقطة توقف هاردوير عند العنوان addr والأنماط الممكنة هي: ▪ "r" قراءة ▪ "w" كتابة ▪ "x" تنفيذ (افتراضي)
	مثـــال	bphws 64A000, "w"
BPL		
	الصيغة	BPL addr, expr
	المفعول	وضع نقطة توقف عند العنوان addr لتدوين العبارة expr
	مثـــال	bpl 64A000, "eax" تدوين قيمة eax عند كل توقف على العنوان 64A000
BPLCND		
	الصيغة	BPLCND addr, expr, cond
	المفعول	وضع نقطة توقف مشروطة بالشرط cond عند العنوان addr لتدوين العبارة expr
	مثـــال	bpl addr, "eax", "ecx ==0" ; تدوين قيمة eax عند كل توقف على العنوان 64A000 ; و عند استيفاء الشرط ecx=0
BPMC		
	الصيغة	BPMC [addr]
	المفعول	حذف نقطة توقف في الذاكرة عند العنوان addr
BPRM		
	الصيغة	BPRM addr, size
	المفعول	ضع نقطة توقف في الذاكرة عند القراءة على العنوان addr بطول size

	مثـــال			bprm 64A000, 08
BPWM				
	الصيغة	BPWM addr, size		
	المفعول	ضع نقطة توقف في الذاكرة عند الكتابة على العنوان addr بطول size		
	مثـــال			bpwm 64A000, 08
BUF				
	الصيغة	BUF var		
	المفعول	تحول المتغير var (عبارة أو قيمة) إلى مقابله الرقمي		
	مثـــال	ترجع DEDE ترجع #44454445#	; ;	mov str, "DEDE" msg str buf str msg str
CMP				
	الصيغة	CMP dest, src [,size]		
	المفعول	شبيهة بنظيرها في لغة التجميع. تقارن src مع dest بطول size		
	مثـــال			cmp y, x cmp eip, 401000, 4
CMT				
	الصيغة	CMT addr, comment		
	المفعول	ضع تعليق comment عند العنوان addr		
	مثـــال			cmt eip, "This is the OEP"
COB				
	الصيغة	COB		
	المفعول	تواصل تنفيذ السكريبت بعد مصادفة نقطة توقف. تلغي عمل الأمر EOB		
	مثـــال			
COE				
	الصيغة	COE		
	المفعول	تواصل تنفيذ السكريبت بعد مصادفة استثناء. تلغي عمل الأمر EOE		
	مثـــال			
DBH				
	الصيغة	DBH		
	المفعول	إخفاء المنقح		
DBS				
	الصيغة	DBS		
	المفعول	إظهار المنقح. تلغي عمل الأمر DBH		
DEC				
	الصيغة	DEC var		
	المفعول	طرح واحد من المتغير var		
	مثـــال			mov var, B

		dec var msg var ; ترجع A0000000
DIV		
	الصيغة	DIV op1, op2
	المفعول	تقسيم op1 على op2 وحفظ الناتج في op1
	مثـــــال	mov op1, 12. mov op2, 6 ترجع op1 = 2 div op1, op2
DM		
	الصيغة	DM addr, size, file
	المفعول	سكب (dump) محتوى الذاكرة من العنوان addr وبطول size إلى الملف file
	مثـــــال	dm eip, 1000, "c:\dump.bin"
DMA		
	الصيغة	DMA addr, size, file
	المفعول	سكب ثم إضافة محتوى الذاكرة من العنوان addr وبطول size إلى الملف file إذا وجد
	مثـــــال	dma eip, 1000, "c:\dump.bin"
DPE		
	الصيغة	DPE file, ep
	المفعول	سكب محتوى الذاكرة للملف التنفيذي المنقح إلى الملف file وجعل نقطة دخوله ep
	مثـــــال	dpe "c:\dump.exe", eip
EOB		
	الصيغة	EOB label
	المفعول	تحويل تنفيذ السكريبت إلى الجذاذة label عند نقطة التوقف التالية
EOE		
	الصيغة	EOE label
	المفعول	تحويل تنفيذ السكريبت إلى الجذاذة label عند الاستثناء التالي
	مثـــــال	eoe end:
ERUN		
	الصيغة	ERUN
	المفعول	مشابهة للضغط على SHIFT-F9. تنفيذ دون التوقف عند الاستثناءات
ESTI		
	الصيغة	ESTI
	المفعول	مشابهة للضغط على SHIFT-F7.
ESTO		
	الصيغة	ESTO
	المفعول	قديمة لكن مشابهة للأمر ESTI.
	مثـــــال	
EVAL		
	الصيغة	EVAL "string with {var}"

	المفعول	تقييم متغير داخل عبارة. والمتغير يجب أن يعرف بين علامتي تنصيص " ".
	مثــــال	mov x, 1F eval "the value of x is {x}" ; $RESULT= "the value of x is 1F"
EXEC/ENDE		
	الصيغة	EXEC ... ENDE
	المفعول	تنفذ تعليمات لغة التجميع المحصورة بين EXEC وENDE في بيئة البرنامج المنقح. المتغيرات المعرفة بين علامتي تنصيص، تعوض بقيمتها.
	مثــــال	mov x, "eax" mov y, "401000" **exec** xor {x}, {x} ; xor eax,eax add {x}, {y} ; add eax, 401000 **ende**
FILL		
	الصيغة	FILL addr, len, value
	المفعول	ملء الذاكرة عند العنوان addr بالقيمة value وبطول len بايت
	مثــــال	fill 401000, FF, 90 ملئ الذاكرة عند 401000 بـــ 128 تعليمة NOP
FIND		
	الصيغة	FIND addr, expr
	المفعول	للبحث عن القيمة expr ابتداءً من العنوان addr. ترجع عنوان أول مصادفة للقيمة في $RESULT.
	مثــــال	find eip, #6AFFE8# البحث عن أي تعليمة push -1 متبوعة بنداء
FINDCALLS		
	الصيغة	FINDCALLS addr [,name]
	المفعول	للبحث عن جميع النداءات القياسية (dll calls) ابتداءً من العنوان addr ويمكن استعمال المرشح name
	مثــــال	findcalls eip, reg البحث عن النداءات الخاصة بالرجستري
FINDCMD		
	الصيغة	FINDCMD addr, cmdstr
	المفعول	للبحث عن التعليمة cmdstr ابتداءً من العنوان addr. يمكن البحث عن سلسلة تعليمات مع الفصل بينها بفاصلة منقوطة ";".
	مثــــال	findcmd eip, "push 0" ; البحث عن التعليمة push 0 findcmd eip, "cmp al,dl; setne bl; not ebx"
FINDOP		
	الصيغة	FINDOP addr, opcode
	المفعول	البحث في الكود عن opcode ابتداءً من العنوان addr
	مثــــال	findop eip, #33??# ; البحث عن أي تعليمة xor
FINDMEM		
	الصيغة	FINDMEM val [, StartAddr
	المفعول	البحث في الذاكرة عن القيمة val ابتداءً من العنوان StartAddr

	مثـــــال	findmem #6A00#
FREE		
	الصيغة	FREE addr [, size]
	المفعول	تحرير الذاكرة (التي غالباً ما سبق وخصصت بالأمر ALLOC) عند العنوان addr وبطول size وفي حالة عدم تعريف الطول size فسيتم تحرير كامل الذاكرة
	مثـــــال	alloc 100 — تخصيص 256 بايت في صفحة ذاكرة جديدة free $RESULT, FF — تحرير 128 بايت من الذاكرة المخصصة
GBPR		
	الصيغة	GBPR
	المفعول	جلب نوع آخر نقطة توقف 10 نقطة توقف عادية 20 نقطة توقف في الذاكرة 40 نقطة توقف صلبة (هاردوير)
	مثـــــال	GBPR cmp $RESULT, 10 je SelectNormalBP cmp $RESULT, 20 je SelectMemBP cmp $RESULT, 40 je SelectHwBP jmp NextBP
GCI		
	الصيغة	GCI addr, info
	المفعول	ترجع معلومات حول تعليمة لغة التجميع عند العنوان addr المعامل info قد يكون أحد القيم التالية: • COMMAND ترجع الــ opcode الموافق للتعليمة. • DESTINATION ترجع عنوان الوصول في حالة تعليمات القفز، النداء والرجوع ret. • SIZE ترجع طول أو عدد بايتات opcode التعليمة. • TYPE ترجع نوع التعليمة: ○ 10 أمر دفع إلى المكدس push. ○ 20 أمر سحب من المكدس pop. ○ 40 أمر fpu. ○ 50 أمر قفزة غير مشروطة. ○ 60 أمر قفزة مشروطة. ○ 70 أمر نداء. ○ 80 أمر رجوع.
	مثـــــال	gci 401000, destination
GCMT		
	الصيغة	GCMT addr
	المفعول	ترجع التعليق عند العنوان addr

	مثـــال	gcmt 40D1F5
GMA		
	الصيغة	GMA name, info
	المفعول	ترجع معلومات عن الوحدة (module) ذات الاسم name المعامل info يمكن أن يأخذ واحدة من القيم التالية: MODULEBASE, MODULESIZE, CODEBASE, CODESIZE, MEMBASE, MEMSIZE, ENTRY, NSECT, DATABASE, RELOCTABLE, RELOCSIZE, RESBASE, RESSIZE, IDATABASE, IDATATABLE, EDATATABLE, EDATASIZE
	مثـــال	gma "crackme", codebase msg $RESULT
GMEMI		
	الصيغة	GMEMI addr, info
	المفعول	ترجع معلومات عن كتلة الذاكرة التي ينتمي لها العنوان addr المعامل info يمكن أن يأخذ واحدة من القيم التالية: MEMORYBASE, MEMORYSIZE, MEMORYOWNER
	مثـــال	gmemi eip, memorysize msg $RESULT
GMI		
	الصيغة	GMI addr, info
	المفعول	ترجع معلومات عن الوحدة (module) التي ينتمي لها العنوان addr المعامل info يمكن أن يأخذ واحدة من القيم التالية: MODULEBASE, MODULESIZE, CODEBASE, CODESIZE, MEMBASE, MEMSIZE, ENTRY, NSECT, DATABASE, RELOCTABLE, RELOCSIZE, RESBASE, RESSIZE, IDATABASE, IDATATABLE, EDATATABLE, EDATASIZE, NAME, PATH, VERSION
	مثـــال	gmi eip, name — جلب اسم الوحدة المنقحة حاليا gma $RESULT, entry — جلب نقطة الدخول msg $RESULT — إظهار نقطة الدخول في رسالة
GN		
	الصيغة	GN addr
	المفعول	▪ ترجع الاسم الرمزي للعنوان addr في $RETURN. ▪ إذا كان العنوان هو لدالة API، ترجع اسم المكتبة (مثال: kernel32) في $RETURN_1 واسم الدالة (مثال: ExitProcess) في $RETURN_2.
	مثـــال	gn eip msg $RESULT
GO		
	الصيغة	GO addr
	المفعول	تنفذ البرنامج حتى العنوان addr
	مثـــال	go eip + 1000
GPA		
	الصيغة	GPA proc, lib
	المفعول	▪ ترجع عنوان الدالة proc داخل المكتبة lib في $RETURN. ▪ ترجع صفر، $RETURN == 0 في حالة عدم وجود الدالة أو المكتبة.

		■ مفيدة عند الحاجة لوضع نقاط توقف على دوال API.	
	مثـــــال	gpi "CreateFileW", "kernel32.dll"	
GPI			
	الصيغة	GPI key	
	المفعول	ترجع معلومات عن الـــ process المنقح. البارومتر key يمكن أن يأخذ واحدة من القيم التالية: HMAINTHREAD, PROCESSID, HPROCESS, ,MAINTHREADID, ,CURRENTDIR EXEFILENAME, PROCESSNAME, MAINBASE, SYSTEMDIR	
	مثـــــال	ترجع الـــ PID الحالي	gpi PROCESSID
GREF			
	الصيغة	GREF [instance]	
	المفعول	■ ترجع العنوان التالي لنتيجة البحث، حيث instance تدل على رقم النتيجة التالية. ■ مفيدة عند الحاجة لإيجاد أكثر من نتيجة بحث.	
	مثـــــال	تهيئ الملف الذي سيكتب فيه الناتج (حذف محتوى السابق إن وجد)	wrt "found.txt", " "
		تهيئة العداد	mov instance,1
		بحث عن التعليمة	find eip, "xor al,al"
			next:
			cmp $RESULT, 0
		إن لم توجد >> خروج	je **finish**
		جلب العنوان التالي	gref instance
		كتابة العنوان الحالي في الملف النصي (واحد بالسطر)	wrta "found.txt", $RESULT,
		الانتقال إلى نتيجة البحث التالي	inc instance
			jmp **next**
			finish: msg "check the file found.txt for search results" ret
INC			
	الصيغة	INC var	
	المفعول	تضيف واحد للمتغير var.	
	مثـــــال	تعرف x وتعطيه القيمة صفر	mov x, 0
		تزيد واحد	inc x
ITOA			
	الصيغة	ITOA n [, base=16]	
	المفعول	تحول القيمة العددية n إلى عبارة عددية في الأساس base (افتراضياً ستة عشري)	
	مثـــــال		mov y, 01F2 itoa y

الأمر	البند	المحتوى	
		mov x, "01F2 equivalent in HEX is: " + $RESULT msg x itoa y,A mov x, "01F2 equivalent in Dec is: " + $RESULT msg x itoa y,2 mov x, "01F2 equivalent in BIN is: " + $RESULT msg x ret	
JA; JAE; JB; JBE; JE; JNE; JMP			
	الصيغة	JMP label	
	المفعول	تقفز إلى الجذاذة label مشابهة لنظيراتها في لغة التجميع.	
	مثـــال	Loop: ... findcmd eip, "setne al"	تبحث عن التعليمة
		cmp $RESULT, 0 je Finish jmp Loop	إذا لم توجد تقفز إلى الجذاذة Finish إذا وجدت تقفز إلى الجذاذة Loop
		Finish: msg "Sorry, Nothing found" ret	
KEY			
	الصيغة	KEY vkcode [, shift [, ctrl]]	
	المفعول	▪ تماثل الضغط على زر ذو الرمز vkode. ▪ يمكن محاكاة الضغط على الزرين shift و ctrl بإعطائهما القيمة 0 أو 1 ▪ للاطلاع على المزيد من vcodes يرجى مراجعة الملحق.	
	مثـــال	shift + F9/ key 78,1 ctrl + space / key 20, 0,1	
LBL			
	الصيغة	LBL addr, text	
	المفعول	تضيف جذاذة عند العنوان addr.	
	مثـــال	lbl eip, "This is the OEP"	
LC			
	الصيغة	LC	
	المفعول	تمسح محتوى نافذة المدونة Log window.	
	مثـــال		
LCLR			
	الصيغة	LCLR	

	المفعول	تمسح محتوى نافذة المدونة الخاصة بالسكريبت.	
	مثـــــال		
LEN			
	الصيغة	LEN str	
	المفعول	ترجع طول العبارة str.	
	مثـــــال		len "AT4RE is my team" msg $RESULT
LM			
	الصيغة	LM addr, size, filename	
	المفعول	▪ تحمل ملف الإفراغ dump ذو الاسم filename في العنوان addr وبطول size. ▪ تعكس عمل الأمر DM.	
	مثـــــال		lm 40100, 1000, "dump.bin"
LOG			
	الصيغة	LOG src [,prefix]	
	المفعول	▪ تدون القيمة src في نافذة المدونة OllyDbg Log Window ▪ إذا كانت src تمثل عبارة ثابتة، فتلك العبارة ستدون كما هي. ▪ إذا كانت src تمثل متغير أو مسجل، فسيتم تدوين قيمة المتغير أو المسجل.	
	مثـــــال		mov x, 1E mov eax, x
		تدون العبارة بين ""	log "AT4RE is the best"
		تدون "x: 0000001E"	log x
		تدون "eax: 0000001E"	log eax
		تدون " 0000001E"	log eax,""
MOV			
	الصيغة	MOV dest, src [,size]	
	المفعول	▪ كمثيلتها في لغة التجميع. ▪ تنقل القيمة src إلى المتغير dest وبطول size. ▪ src قد تكون عبارة نصية، عبارة ستة عشرية، قيمة عددية، مسجل أو راية.	
	مثـــــال		mov x, 0F
		نقل عبارة نصية إلى متغير	mov y, "Hello world"
		نقل مسجل إلى مسجل	mov eax, ecx
		نقل عبارة ستة عشرية إلى العنوان المؤشر بـــ ecx	mov [ecx], #DEADC0DE#
		تفعيل راية الاحتفاظ	mov !CF, 1
		نقل حالة راية الازدواجية إلى راية الاتجاه	mov !DF, !
		نقل العبارة النصية إلى العنوان 403000	mov [403000], "Hello world"
MEMCP			

Y	الصيغة	MEMCPY dest,src,size		
	المفعول	▪ تنقل القيمة المؤشر لها بالعنوان src إلى العنوان dest وبطول size. ▪ مطابقة للأمر : mov [dst],[src],size.		
	مثـــال	نقل محتوى الذاكرة انطلاقا من Code Base إلى المساحة المخصصة وبطول Code Size		
			جلب Code Base للوحدة GDI32.dll	gma "GDI32",CODEBASE
			نقله إلى المتغير	mov base, $RESULT
			جلب Code Size للوحدة GDI32.dll	gma "GDI32",CODESIZE
			نقله إلى المتغير	mov size, $RESULT
			تخصيص مساحة في الذاكرة بطول size	alloc size
			نقل العنوان المخصص إلى المتغير	mov dst,$RESULT
				memcpy dst,base,size
MSG				
	الصيغة	MSG message		
	المفعول	و تظهر رسالة بمضمون message		
	مثـــال	msg "OEP Found!"		
MSGYN				
	الصيغة	MSGYN message		
	المفعول	تظهر رسالة بمضمون message مع زرين "YES" و"NO" ترجع 0 إذا ضغط الزر "NO" و 1 إذا ضغط الزر "YES"		
	مثـــال			msgyn "Do you want to exit?" cmp $Result,0 je Continue Exit: msg "Script terminated by user" ret Continue: .. ..
MUL				
	الصيغة	MUL op1, op2		
	المفعول	كمثيلتها بلغة التجميع، تحفظ في المعامل op1 ناتج ضرب op1 بـ op2		
	مثـــال	mov x, 0F mul x, 2		
NEG				
	الصيغة	NEG op		
	المفعول	كمثيلتها بلغة التجميع، تحسب المكمل الثنائي (Two's Complement) للمعامل op		
	مثـــال			mov x, 15 neg x

		eval "the 2's complement of 15 is {x}" msg $RESULT ret	ترجع "FFFFFFEB"
NOT			
	الصيغة	الصيغة: NOT op	
	المفعول	كمثيلتها بلغة التجميع، تحسب المكمل الأحادي (One's Complement) للمعامل op	
	مثـــال	mov x, 15 not x eval "the 1's complement of 15 is {x}" msg $RESULT ret	ترجع "FFFFFFEA"
OR			
	الصيغة	OR dest, src	
	المفعول	كمثيلتها بلغة التجميع، تقوم بالعملية المنطقية "OR" بين dest و src، تحفظ الناتج في dest	
	مثـــال	mov y, A2 mov x, 6B or x, y eval "x OR y = {x}" ret	ترجع "EB000000"
OPCODE			
	الصيغة	OPCODE addr	
	المفعول	▪ ترجع في $RESULT الــ Opcode المقابل للتعليمة الموجودة على العنوان addr. ▪ ترجع في $RESULT_2 طول الــ Opcode. ▪ في حال Opcode خاطئ، ترجع $RESULT_2 = 0.	
	مثـــال	opcode eip msg $RESULT msg $RESULT_2	opcode عند eip طول الكود
OPENDUMP			
	الصيغة	OPENDUMP addr [,base,size]	
	المفعول	تفتح نافذة سكب (dump) لمحتوى الذاكرة عند العنوان addr أو ابتداءً من العنوان base وبطول size	
	مثـــال	opendump eip opendump eip, eip, 1000	تفتح نافذة سكب للذاكرة عند العنوان eip تفتح نافذة سكب للذاكرة عند العنوان eip بطول h1000
OPENTRACE			
	الصيغة	OPENTRACE	
	المفعول	تفتح نافذة التعقب (trace window)	
PAUSE			
	الصيغة	PAUSE	
	المفعول	▪ تعلق تنفيذ السكريبت. ▪ لمواصلة التنفيذ يجب النقر على الأمر Resume من القائمة المنسدلة للإضافة OdbgScript.	

POP	الصيغة	POP dword	
	المفعول	كمثيلتها بلغة التجميع، تقوم بإخراج أول قيمة في المكسس إلى المتغير dword	
	مثــــال	أول قيمة في المكدس إلى eax	pop eax
		القيمة التالية في المكدس إلى العنوان C00058	pop [58C000]
PREOP	الصيغة	PREOP addr	
	المفعول	▪ ترجع عنوان التعليمة السابقة، للتعليمة الموجودة على العنوان addr. ▪ تنبيه: لا ترجع بالضرورة آخر تعليمة نفذت، أي لن ترجع مثلا عنوان قفزة مستوفاة. ▪ بمعنى آخر، PREOP ترجع فقط عنوان التعليمة في السطر قبل الحالي.	
	مثــــال		preop eip
PUSH	الصيغة	PUSH dword	
	المفعول	كمثيلتها بلغة التجميع، تقوم بدفع قيمة المتغير dword إلى المكدس	
	مثــــال	eax إلى المكدس	push eax
		القيمة في العنوان C00058 إلى المكدس	push [58C000]
READSTR	الصيغة	READSTR str, len	
	المفعول	ترجع عدد len من الأحرف من العبارة str ابتداءً من اليسار	
	مثــــال	ترجع "AT4RE"	mov str, "AT4RE is the best" readstr str, 5 msg $RESULT
REF	الصيغة	REF addr	
	المفعول	▪ شبيهة بالضغط على CTRL + R أو خاصية Find references to. ▪ ترجع في $RESULT أول مرجع (reference) للعنوان addr. ▪ ترجع في $RESULT_1 التعليمة عند المرجع. ▪ ترجع في $RESULT_2 التعليق عند المرجع. ▪ لجلب باقي المراجع يعاد تنفيذ الأمر REF إلى حين RESULT = 0$.	
	مثــــال		**Repeat**: ref eip log $RESULT log $RESULT_1 log $RESULT_2 cmp $RESULT, 0 jne **Repeat** ret
REPL	الصيغة	REPL addr, find, repl, len	
	المفعول	تبدل العبارة الرقمية find بالعبارة الرقمية repl ابتداءً من العنوان addr وعلى طول len بايت	

	مثـــال	تبدل push 0 بـــ push 1 تلغي أي قفزة jnz تبدل inc eax بـــ nop	repl eip, #6A00#, #6A01#, 10 repl eip, #75??#, #7500#, 1F repl 40100, #40#, #90#, 1F
RET			
	الصيغة	RET	
	المفعول	تنهي تنفيذ السكريبت أو ترجع من مناداة دالة	
REV			
	الصيغة	REV dword	
	المفعول	ترجع القيمة المعكوسة للقيمة dword	
	مثـــال	ترجع 67452301	rev 01234567 msg $RESULT
ROL			
	الصيغة	ROL op, count	
	المفعول	كمثيلتها بلغة التجميع، تقوم بتدوير القيمة op إلى اليسار عدد count بت، القيمة الناتجة في op	
	مثـــال	ترجع 23456781	rol eax, cl rol [58C000], 10 rol 12345678, 4
ROR			
	الصيغة	ROR op, count	
	المفعول	كمثيلتها بلغة التجميع، تقوم بتدوير القيمة op إلى اليمين عدد count بت، القيمة الناتجة في op	
	مثـــال	ترجع 81234567	ror eax, cl ror [58C000], 10 ror 12345678, 4
RTR			
	الصيغة	RTR	
	المفعول	شبيهة بالضغط على CTRL + F9 أو تنفيذ العملية Run till return في OllyDbg	
RTU			
	الصيغة	RTU	
	المفعول	شبيهة بالضغط على ALT + F9 أو تنفيذ العملية Run till user code في OllyDbg	
RUN			
	الصيغة	RUN	
	المفعول	شبيهة بالضغط على F9 أو تنفيذ العملية Run في OllyDbg	
SCMPI			
	الصيغة	SCMPI dest, src [,size]	
	المفعول	شبيهة بالأمر CMP مع فارق بسيط: SCMPI لا تعير اهتماماً لكبر أو صغر الحروف المثال التالي دائما يرجع عبارة "Good"	
	مثـــال		mov x, "at4re" scmpi x, "AT4RE" je **Good** msg "Bad" **Good**:

				msg "Good" ret
SETOPTION				
	الصيغة	SETOPTION		
	المفعول	تفتح نافذة الخيارات في OllyDbg		
SHL				
	الصيغة	SHL op, count		
	المفعول	كمثيلتها بلغة التجميع، تقوم بإزاحة القيمة op إلى اليسارعدد count بت، القيمة الناتجة في op		
	مثـــــال		ترجع 23456780	hl eax, cl shl [58C000], 10 mov x,12345678 shl x, 4
SHR				
	الصيغة	SHR op, count		
	المفعول	كمثيلتها بلغة التجميع، تقوم بإزاحة القيمة op إلى اليسارعدد count بت، القيمة الناتجة في op		
	مثـــــال		ترجع 01234567	hr eax, cl shr [58C000], 10 mov x,12345678 shr x, 4
STI				
	الصيغة	STI		
	المفعول	شبيهة بالضغط على F7 أو تنفيذ العملية Step Into في OllyDbg		
STO				
	الصيغة	STO		
	المفعول	شبيهة بالضغط على F8 أو تنفيذ العملية Step Over في OllyDbg		
STR				
	الصيغة	STR var		
	المفعول	عكس BUF، تحول المتغير var (قيمة عددية) إلى مقابله النصي		
	مثـــــال		ترجع AT4RE	mov strng, "AT4RE" msg strng buf strng
			ترجع #4154345245#	msg strng str strng
			ترجع AT4RE	msg strng
SUB				
	الصيغة	SUB dest, src		
	المفعول	كمثيلتها بلغة التجميع، تقوم بطرح قيمة src من dest وتحفظ الناتج في dest		
	مثـــــال			sub eax, ebx sub [58C000], 1000
TC				
	الصيغة	TC		
	المفعول	تقوم بإيقاف التعقب (Tracing) في OllyDbg		

TEST

الصيغة: TEST dest, src

المفعول: كمثيلتها بلغة التجميع، تقوم بعملية AND بين dest و src دون حفظ الناتج.
تغير قيمة الرايات CF OF PF SF ZF حسب الناتج.

مثـــال:

```
ask "Input x:"
cmp $RESULT, 0
je canceled
mov x, $RESULT
ask "Input y:"
cmp $RESULT, 0
je canceled
mov y, $RESULT
test x, y
eval "flags after TEST x,y : CF= {!CF} OF= {!OF} PF= {!PF}
SF= {!SF} ZF= {!ZF}"
msg $RESULT
canceled:
ret
```

TI

الصيغة: TI

المفعول: شبيهة بالضغط على CTRL + F11 أو تنفيذ العملية Trane Into في OllyDbg

TICK

الصيغة: TICK var

المفعول: ترجع في المتغير var وقت تنفيذ السكريبت منذ بدايته بألف جزء من الثانية ms

مثـــال:

```
tick var
itoa var,10.          تحويل إلى النظام العشري
eval "script took {$RESULT} ms"
msg $RESULT
```

TICND

الصيغة: TICND cond

المفعول: تتعقب داخل النداءات (Trace Into) إلى حين تحقق الشرط cond

مثـــال:

```
ticnd "eip > 402000"
opentrace
ret
```

TO

الصيغة: TO

المفعول: شبيهة بالضغط على CTRL + F12 أو تنفيذ العملية Trane Over في OllyDbg

TOCND

الصيغة: TOCND cond

المفعول: تتعقب خارج النداءات (Trace Over) إلى حين تحقق الشرط cond

مثـــال:

```
tocnd "eip > 402000"
opentrace
ret
```

VAR

الصيغة: VAR var

	المفعول	تعلن المتغير var دون إعطاءه قيمة الأمر MOV يقوم بالمثل مع إعطاء قيمة للمتغير
XOR		
	الصيغة	XOR dest, src
	المفعول	كمثيلتها بلغة التجميع، تقوم بعملية xor بين src و dest وتحفظ الناتج في dest
	مثـــال	xor eax, ebx xor [58C000], 1F
XCHG		
	الصيغة	XCHG dest, src
	المفعول	تقوم بنقل src إلى dest و dest إلى src
	مثـــال	xchg eax, ebx xchg [58C000], 1F xchg [58C100], [58C200]
WRT		
	الصيغة	WRT file, data [,separator]
	المفعول	تنشأ الملف file تم تكتب فيه المعلومة data وتضيف الفاصل separator في الأخير في حال وجود الملف، يتم مسح محتواه
WRTA		
	الصيغة	WRTA file, data [,separator]
	المفعول	تكتب في الملف file المعلومة data وتضيف الفاصل separator في الأخير في حال وجود الملف، يتم إكمال محتواه
	مثـــال	انظر إلى التطبيق العملي في الملحق

9. ملحق

9.1 تطبيق عملي

يمكن تحميل ملف التطبيق ومثال من السكريبت والملف الناتج من هنا.

في هذا المثال سنحاول استخلاص سريالات تسجيل البرنامج المرفق والتي هي لحسن لحظ محقونة في كود البرنامج!

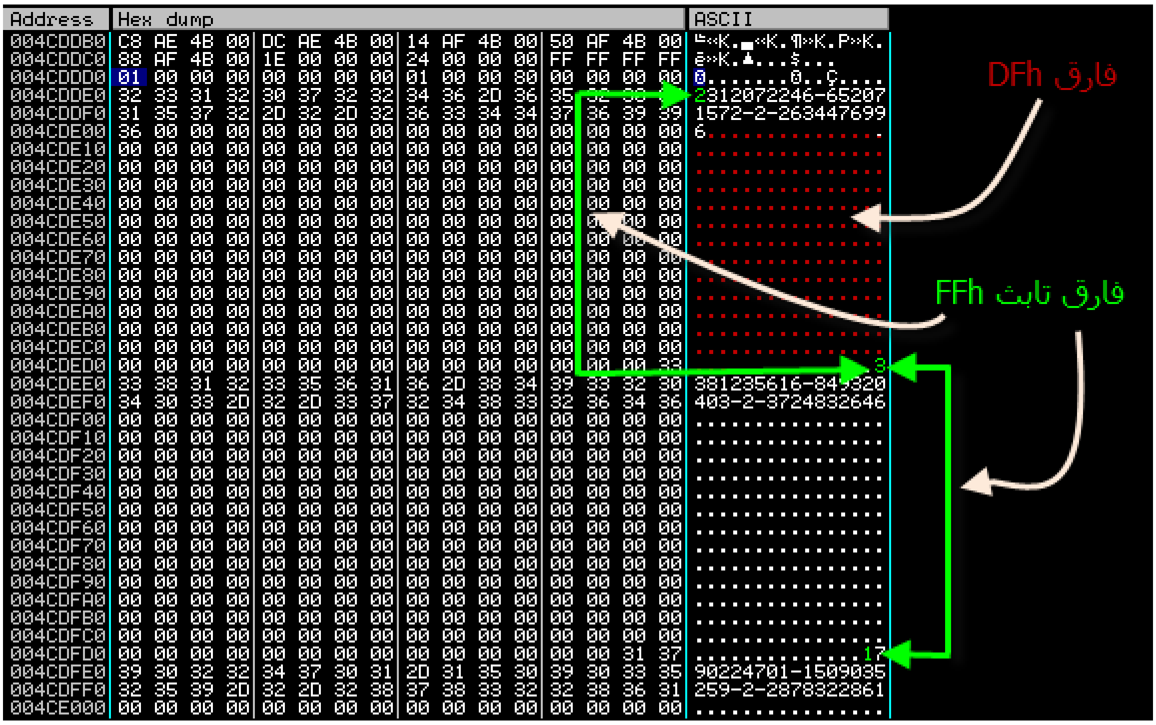
البرنامج يستعملها من أجل المقارنة مع ما يدخله المستخدم حين التسجيل، يعني أنه لا وجود لخوارزمية حساب لسريال صحيح.

إذن كل ما علينا فعله هو صيد هذه السريالات وتجميعها وإن شئنا استعملناها في مولد كيجين عشوائي.

لكن ليس الأمر بالهين فالسريالات شبه مبعثرة - كثيرة وطويلة- مما يجعل جمعها يدوياً أمراً شاقاً.

هنا تأتي فائدة وثمرة تطبيق ما تعلمناه عن OdbgScript وجعل أمر جمع السريالات يتم آليا وفي وقت وجيز.

لنلقي أولا نظرة عن كيفية تواجد السريالات في البرنامج , نذهب إلى العنوان 4CDDD0 وسرجد ما يلي :



هدفنا هنا هو إيجاد منظومة تحكم تراصف السريالات لكي نستغلها في كتابة سكريبت يقوم بجمعها.

ملاحظة أولى : السريالات متباعدة تارة بـــ DFh خانة، وتارة بـــ DEh خانة فيما بينها، وحتى بـــ DDh في أماكن أخرى، وهذا ليس بالجيد لأنه لا يدل على الترتيب والتنظيم.

ملاحظة ثانية : أطوال السريالات تختلف عشوائياً، تارة 31 خانة، وتارة 32 خانة وأحياناً أخرى 33 خانة.

لكن إذا أمعنا النظر سرجد أن اختلافَ الفرق في التباعد بين كل سريال والذي يليه متناسبٌ مع اختلاف طول كل واحدٍ منهما، وبالتالي نستخلص أن التباعد بين أول خانة في كل سريال وأول خانة للسريال الذي يليه هي قيمة ثابتة وتساوي FFh خانة.

لنستثمر هذه الخاصية في كتابة سكريبت:

بداية سنقوم بتعريف متغيرين اثنين، الأول نسميه addr ونعطيه عنوان أول سريال حتى نبدأ العد من هناك.

والثاني نسميه instance والذي سيلعب دور العداد لعدد السريالات التي سنجلها :

```
mov addr, 04CDDE0
mov instance, 0
```

نقوم بعدها بإنشاء ملف نصي نسميه serials.txt نضع فيه السريالات المصطادة، ونكتب فيه كقيمة أولية النص: "Valid Serials:".

سنستعمل الأمر WRT لتدمير محتوى أي ملف يحمل نفس الاسم إن كان موجوداً:

wrt "serials.txt", "Valid Serials:"

الآن نقوم بإنشاء الحلقة التي ستقف عند بداية كل سريال لتكتبه في ملفنا المذكور سابقاً :

```
loop:
   cmp [addr], 0
   je end
   wrta "serials.txt", [addr]
   add addr, FF
   inc instance
jmp loop
```

ثم نظهر الناتج بعد تحويله إلى النظام العشري :

```
end:
   itoa instance, 0A
   eval "{$RESULT} serials were fished"
   msg $RESULT
ret
```

ننفذ السكريبت في OllyDbg لنحصل بعدها على ملف serials.txt (نجده بجانب ملف البرنامج: المسار الافتراضي) وبه كل السريالات منظمة.

9.2. Vcodes

	اضغط الصورة لتحميل المرفقات.

فك حماية PC-GUARD for Win32 الإصدار رقم: 5.00



I. المقدمة

هذا البرنامج يدخل في عائلة برامج الـــ Licensing ويقصد بذلك أننا باستعمالنا لهذا البرنامج لن نحصل فقط على برنامج محمي ضد الهندسة العكسية بل يمكننا عن طريقه إدماج آلية تسجيل كاملة ومتنوعة حسب الحاجة. الحصول على برنامج قابل للتسجيل ومحدود في الزمن والاستعمال..

لكي لا نتطرق إلى شرح كل خصائص البرنامج يمكنك أخي القارئ تجربة النسخة المرفقة والتطلع على كل خصائصها لأننا سنحاول في هذا الدرس إن شاء الله إعطاء توضيح عن كيفية فك حماية البرنامج باستغلال الـــ Exceptions الناتجة عن التتبع بالمنقح.

وقبل البداية يمكننا القول: إن عدم كفاءة هذا النوع من الحماية مع استعمال خاصية الـــ Licensing مشروط ببقاء الـــ Loader يعني فك أي برنامج محمي بهذه الطريقة المذكورة سابقاً يجعله كاملاً دون أي تحديدات في الاستعمال.

- PEiD للفحص أو AT4RE FastScanner.
- OllyDbg.
- OllyDump Plugin لعمل نسخة من الذاكرة.
- Import REConstructor إعادة بناء الدوال المستدعاة من البرنامج.

هذا الدرس موجه إلى المستوى الشبه متوسط، وهو مدعوم بالصور ليكون أدعى للفهم إن شاء الله.

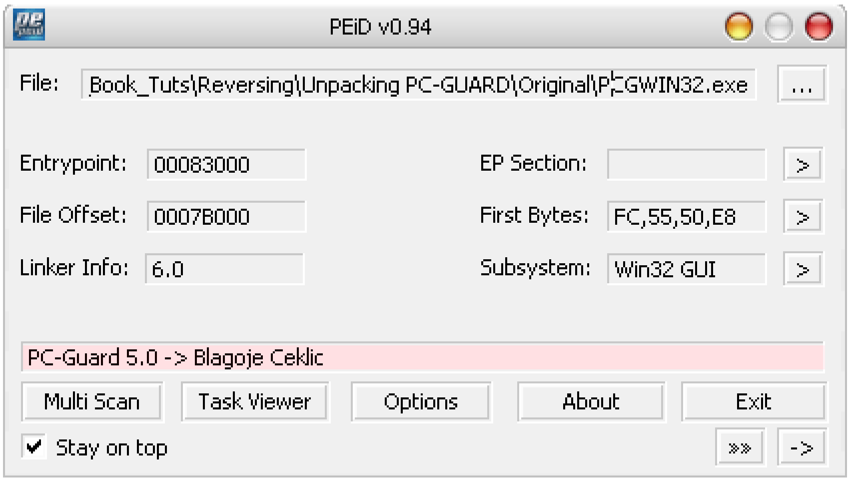
تم تقسيم الدرس إلى قسمين :

- فك حماية الملف التنفيذي PCGWIN32.exe
- فك حماية ملف المكتبة الديناميكية PCGW32.DLL

2. فك حماية الملف التنفيذي PCGWIN32.exe

2.1 فحص البرنامج

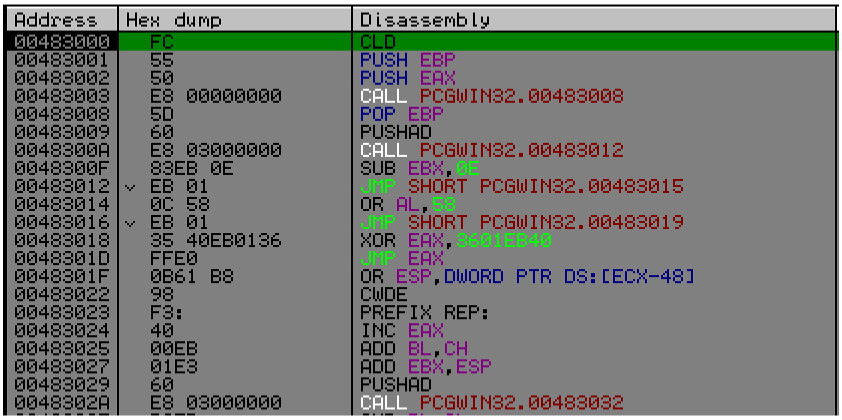
استعمال برامج فحص مثل PEiD أو AT4RE FastScanner للتأكد من وجود حماية أو ضغط أو معرفة اللغة المستعملة في البرمجة أمر أساسي في عالم الهندسة العكسية.



نلاحظ أن البرنامج مضغوط ومشفر بواسطة برنامج PC-Guard 5.0 ويمكن التأكد من المعلومات المعطاة من طرف أداة الفحص من خلال متابعة البرنامج في المنقح لتفادي الوقوع في أخطاء سببها أدوات صممت لتزوير بصمات البرامج مثل AT4RE. Signs_Imitator

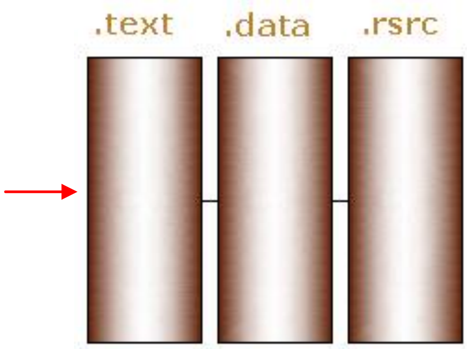
2.2 الـ Loader

نحمل البرنامج في المنقح وبعدها سنلاحظ نقطة دخول البرنامج الغير حقيقية 00483000 وهي بداية الـــ Loader المسؤول عن فك الحماية.



Address	Hex dump	Disassembly
00483000	FC	CLD
00483001	55	PUSH EBP
00483002	50	PUSH EAX
00483003	E8 00000000	CALL PCGWIN32.00483008
00483008	5D	POP EBP
00483009	60	PUSHAD
0048300A	E8 03000000	CALL PCGWIN32.00483012
0048300F	83EB 0E	SUB EBX,0E
00483012	v EB 01	JMP SHORT PCGWIN32.00483015
00483014	0C 58	OR AL,58
00483016	v EB 01	JMP SHORT PCGWIN32.00483019
00483018	35 40EB0136	XOR EAX,3601EB40
0048301D	FFE0	JMP EAX
0048301F	0B61 B8	OR ESP,DWORD PTR DS:[ECX-48]
00483022	98	CWDE
00483023	F3:	PREFIX REP:
00483024	40	INC EAX
00483025	00EB	ADD BL,CH
00483027	01E3	ADD EBX,ESP
00483029	60	PUSHAD
0048302A	E8 03000000	CALL PCGWIN32.00483032

لكي نفهم الفكرة أكثر ونستوعبها بشكلٍ جيدٍ فهذا مثال للأقسام الخاصة بالآلة الحاسبة :

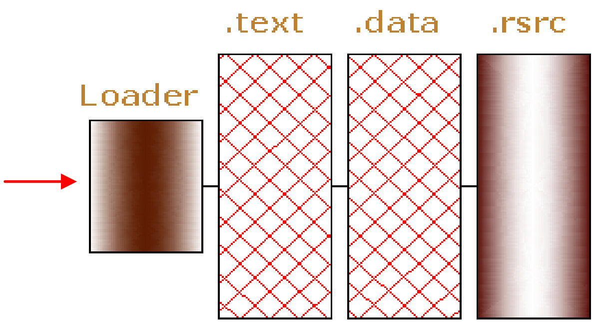


قبل وضع الحماية نقطة دخول البرنامج موجودة في القسم text. ويمكننا قراءة الـــ Strings والاستدعاءات بوضوح في المنقح.

عملية الحماية تمر بمرحلتين :

1. حماية البرنامج الهدف: برنامج الحماية يشفر، يضغط ، يجمع الـــ Loader في البرنامج الهدف ويغير نقطة الدخول الحقيقية من القسم text. إلى القسم الذي يحتوي الـــ Loader.

يصبح الناتج هكذا :



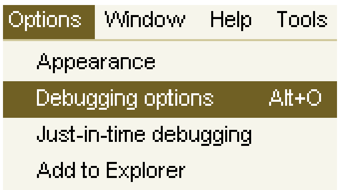
2. الـــ Loader يحتوي على تعليمات فك التشفير والضغط وإرجاع نقطة الدخول الحقيقية، وكل هذه العمليات تتم في الذاكرة، وهو يحتوي أيضا على كل الخدع المعروفة لإيقاف المنقح وتشويش وتشفير الكود لجعل تتبع الـــ Loader أمراً صعباً ومملاً، ويحوي أيضا إجراءات التسجيل والتحديدات الخاصة بزمن ومدة استخدام البرنامج في البرامج التي تحتوي على خاصية Licensing.

قوة برنامج الحماية تكمن في: برمجة Loader صعب التتبع، ويحتوي على خدع تجعل الوصول إلى النقطة الحقيقية أمراً صعباً إضافة إلى جعل إعادة بناء الاستدعاءات شيئاً ليس في متناول الجميع.

2.3 فك الحماية

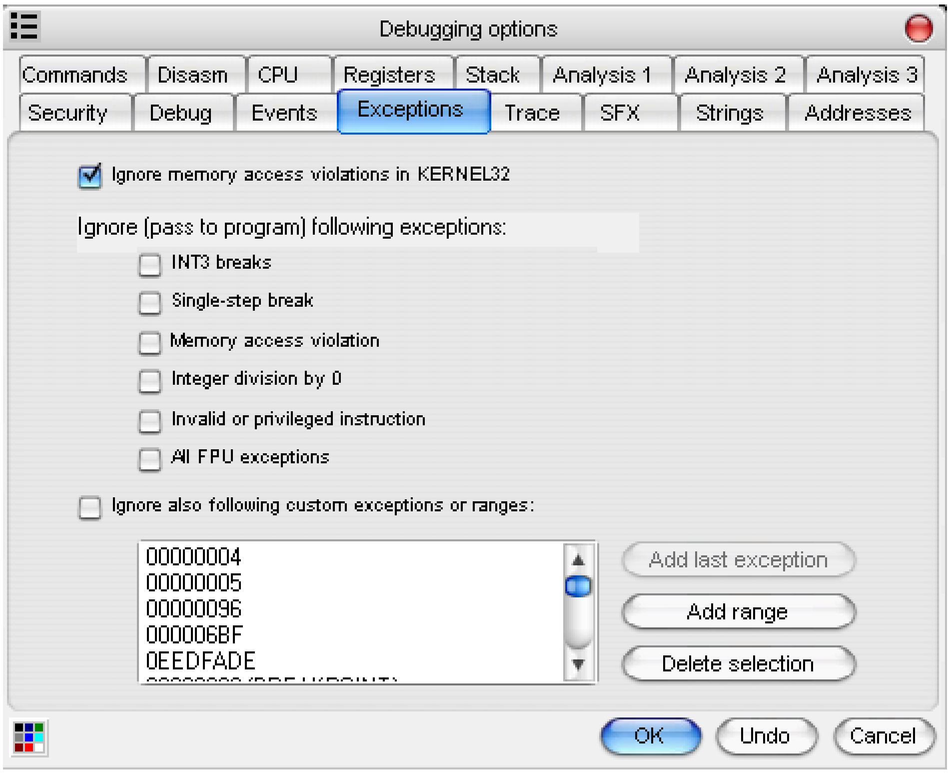
يحتوي هذا الجانب من الدرس الطريقة العملية لفك الحماية.

2.3.1 إعدادات المنقح



نذهب إلى: Options ثم Debugging options

نلغي تفعيل الخيارات كالتالي :

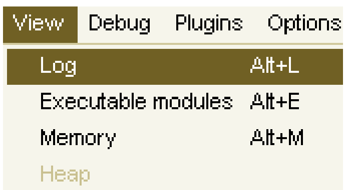


بعد التعديل الجديد نحفظ الخيارات.

> يمكننا تعديل خيارات المنقح كل مرة وحسب نوع الحماية التي نريد دراستها.

2.3.2. نقطة البداية الحقيقية

للحصول على قائمة الـــ Exceptions وفهم ماذا يحدث قبل ظهور نافذة البرنامج؟



نذهب إلى : View ثم Log ثم نحاول تشغيل البرنامج باستعمال Shift F9 لتخطي الـــ Exceptions كما هو موضح في شريط المنقح:

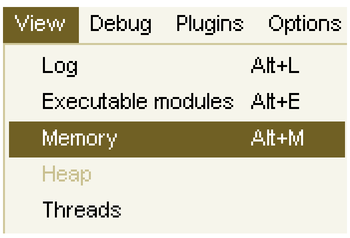


نعيد الكرة إلى أن تظهر نافذة البرنامج.

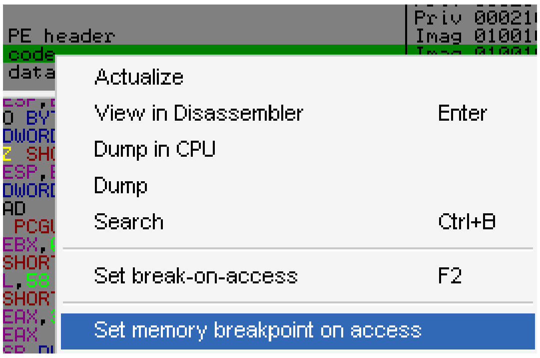
الآن نذهب إلى Log ونحسب عدد الـــ Exceptions الناتجة قبل أن يعمل البرنامج.

	تشغيل البرنامج	F9
00483343	Access violation when writing to [657EAB6E]	Shift + F9
00483564	Access violation when writing to [B4E6507E]	Shift + F9
004836D8	Single step event at PCGWIN32.004836D8	Shift + F9
00483A0D	Single step event at PCGWIN32.00483A0D	Shift + F9
7C911230	INT3 command at ntdll.DbgBreakPoint	
7C911230	INT3 command at ntdll.DbgBreakPoint	
7C911230	INT3 command at ntdll.DbgBreakPoint	
00492999	Single step event at PCGWIN32.00492999	Shift + F9
0049698F	Single step event at PCGWIN32.0049698F	Shift + F9
0046E343	Access violation when writing to [657EAB6E]	Shift + F9
0046E564	Access violation when writing to [B4E6507E]	Shift + F9
0046E6D8	Single step event at PCGWIN32.0046E6D8	Shift + F9
0046EA0D	Single step event at PCGWIN32.0046EA0D	Shift + F9
7C911230	INT3 command at ntdll.DbgBreakPoint	
7C911230	INT3 command at ntdll.DbgBreakPoint	
7C911230	INT3 command at ntdll.DbgBreakPoint	
0047D999	Single step event at PCGWIN32.0047D999	Shift + F9
58B50000	Module C:\WINDOWS\system32\COMCTL32.dll	
76C40000	Module C:\WINDOWS\system32\IMAGEHLP.dll	
76340000	Module C:\WINDOWS\system32\comdlg32.dll	
77F40000	Module C:\WINDOWS\system32\SHLWAPI.dll	
7C9D0000	Module C:\WINDOWS\system32\SHELL32.dll	
77390000	Module C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9\comctl32.dll	
0048198F	Single step event at PCGWIN32.0048198F	Shift + F9
	Set Memory Breakpoint on Access نضع هنا	
77BD0000	Module C:\WINDOWS\system32\VERSION.dll	
5B090000	Module C:\WINDOWS\system32\uxtheme.dll	
	تظهر نافذة البرنامج	

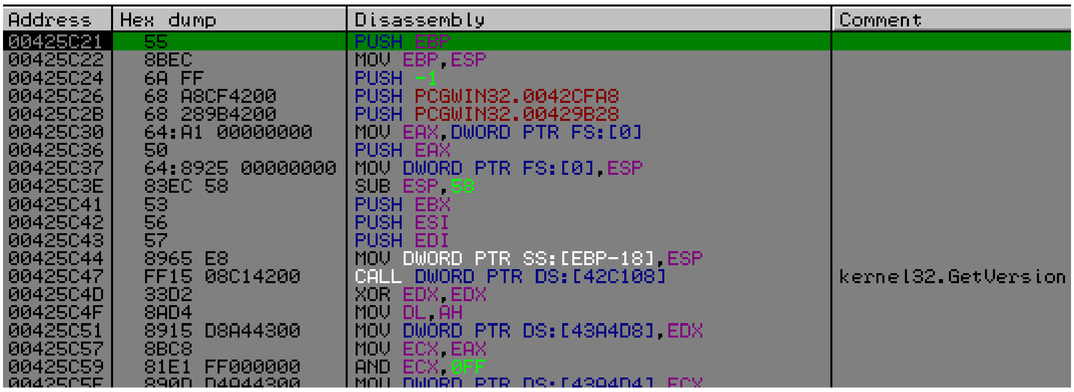
بعد آخر Exception نضع نقطة توقف Set Memory Breakpoint on Access لأنه سيتم الدخول إلى نقطة البداية الحقيقية للبرنامج والخروج من الـــ Loader.



نذهب إلى : View ثم Memory



نختار القسم code ثم نضع نقطة توقف، في حالة الدخول لهذا القسم فإن المنقح سيتوقف.



جميل توقف المنقح في نقطة البداية الحقيقية للبرنامج في الذاكرة.

> لنتأكد أننا في نقطة البداية الحقيقية للبرنامج: نذهب إلى نافذة الـــ Strings و الاستدعاءات، فإذا استطعنا قراءتها بوضوح فالعملية ناجحة.

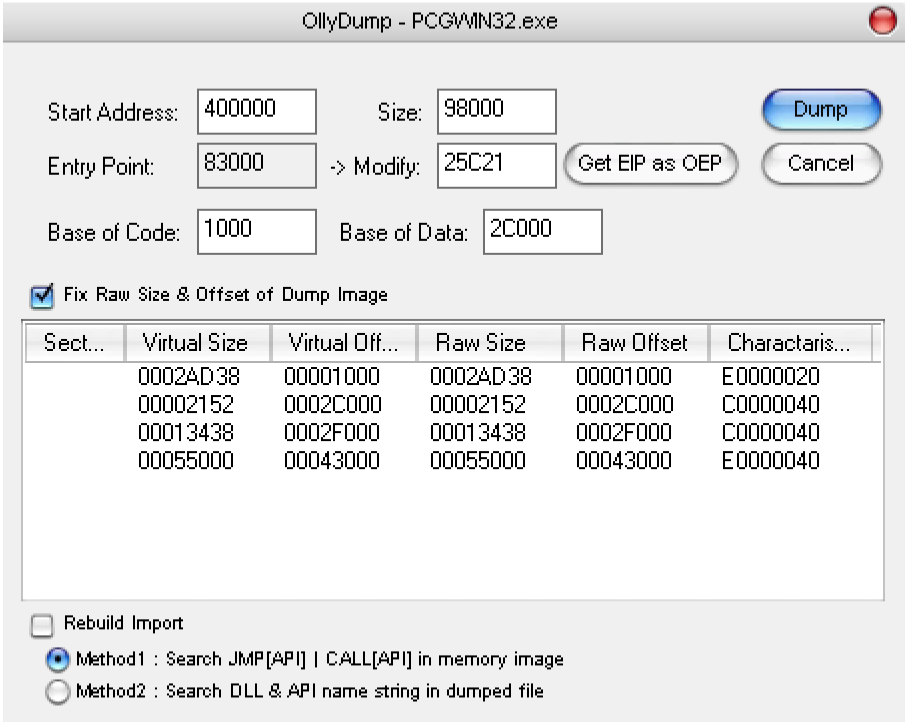
2.3.3. حفظ البرنامج من الذاكرة Dump

سوف نحاول إن شاء الله حفظ البرنامج ابتداء من نقطة البداية الجديدة في ملف جديد من الذاكرة وهذا ما يسمى بعمل Dump

> سنستعمل أداة OllyDump Plugin.
>
> في هذه المرحلة لا يجوز أن نغلق المنقح حتى ننتهي من عملية فك الحماية.



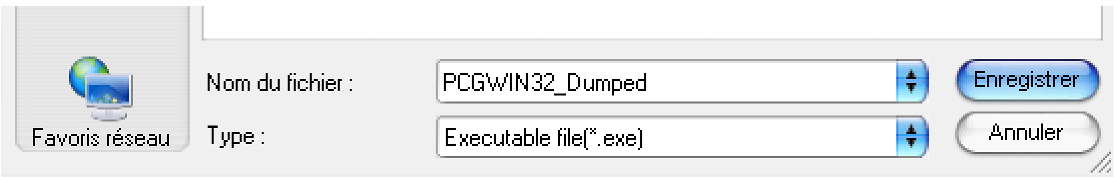
نذهب إلى Plugins ثم OllyDump ثم Dump debugged process





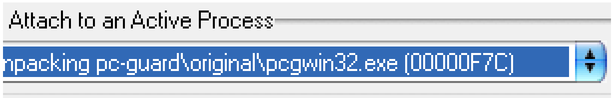
نلغي تفعيل Rebuild import لأننا سنستعمل برنامج :

.Import REConstructor



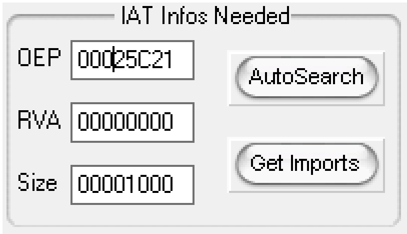
نقوم بحفظ البرنامج الجديد من الذاكرة إلى القرص.

2.3.4. إعادة بناء IMPORTS باستعمال برنامج IMPORT RECONSTRUCTOR



بعد تشغيل أداة إعادة بناء الـــ imports نذهب إلى : Attach to an Active process ونختار برنامج الدرس.

برنامج الدرس محمل في الذاكرة عن طريق المنقح ومتوقف في نقطة البداية الحقيقية، لذا لا يجوز أن نغلق المنقح حتى تنتهي العملية كاملة كما ذكرنا سابقاً.



نذهب إلى IAT Infos Needed لنضع عنوان نقطة البداية الحقيقية المعروفة برمز OEP :



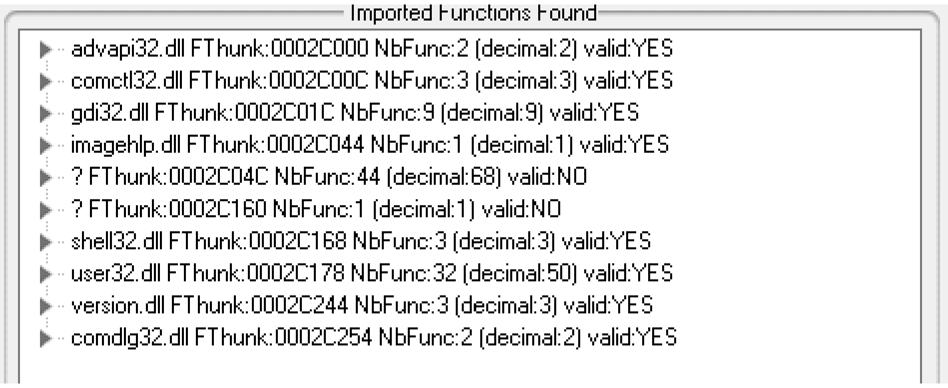
ثم على AutoSearch :



في حالة وجود نتيجة إيجابية تظهر هذه الرسالة

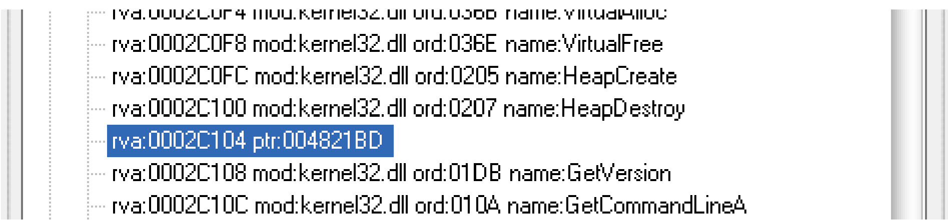


نضغط على Get Imports :

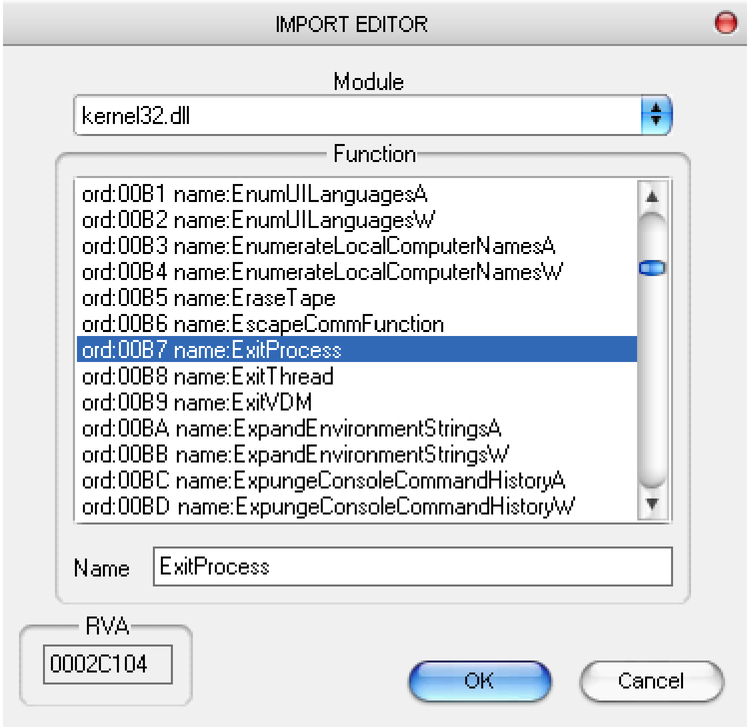


هذه الــ Imports التي تعرف عليها البرنامج، والتي يستطيع تثبيت عناوينها، ولكن قبل التثبيت يجب فحص الــ imports ليتم إصلاحها يدوياً.

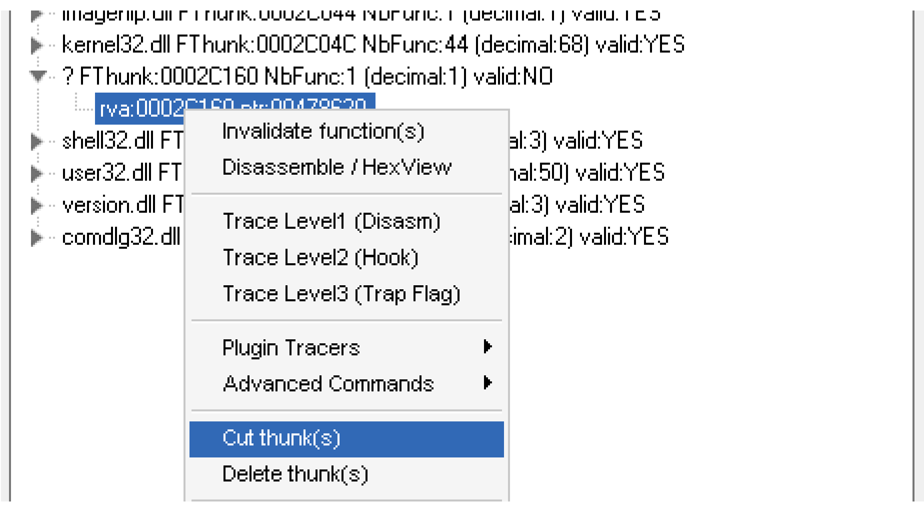
نضغط على Show Invalid 



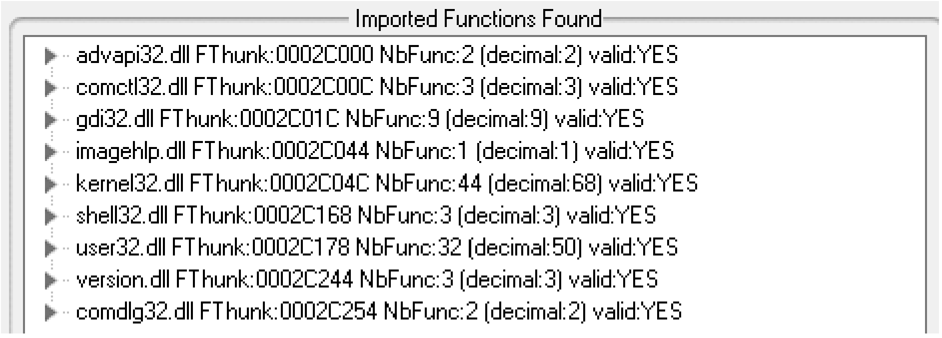
تظهر لنا دالة في المكتبة الديناميكية Kernel32.dll غير معروفة، ونستطيع التعرف عليها بإتباع الترتيب في المنقح لبرنامج الدرس؛ حيث أنها تقع بين دالة HeapDestroy و GetVersion ، وهي دالة ExitProcess التي سوف نصلحها باستعمال خيار : Import Editor



نختار ExitProcess ثم نحفظ الخيار :



بما يخص الـــ Import التالي الغير معروف فلا يوجد لدينا اسم المكتبة ولهذا سنكتفي بحذفها من القائمة ولكن ننتبه لهذه العملية لأنه في بعض الحالات لا يجوز حذف الـــ Import الغير معروفة ومحاولة إصلاحها يدوياً؛ لأن حذفها يؤثر على البرنامج ، نضغط الزر الأيمن للفأرة ثم نختار Cut Thunks.

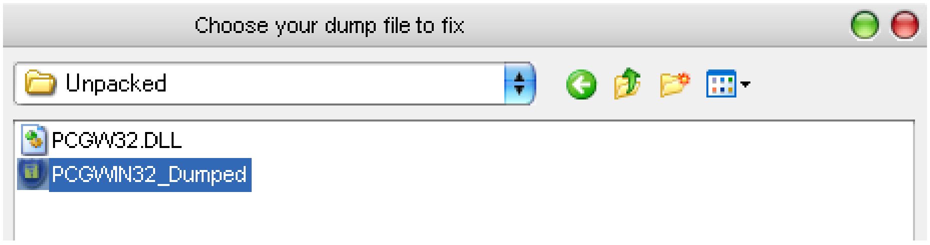


الآن أصبح لدينا عناوين Import صحيحة وسنحاول تثبيتها في البرنامج الجديد Dump .

نضغط على Fix Dump

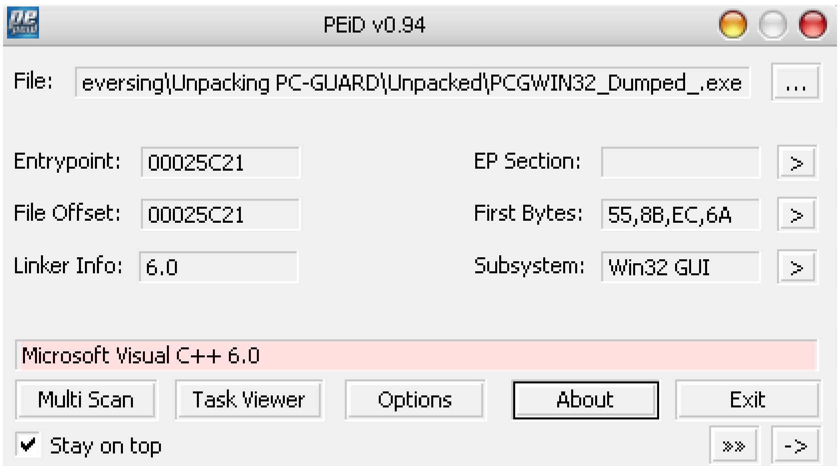


ثم نختار البرنامج الجديد Dump.



بعد إتمام كل مراحل العملية نحاول تشغيل البرنامج والتركيز على تجريب خصائصه.

نقوم بعمل فحص جديد بعد فك الحماية لنشاهد نتيجة العملية :

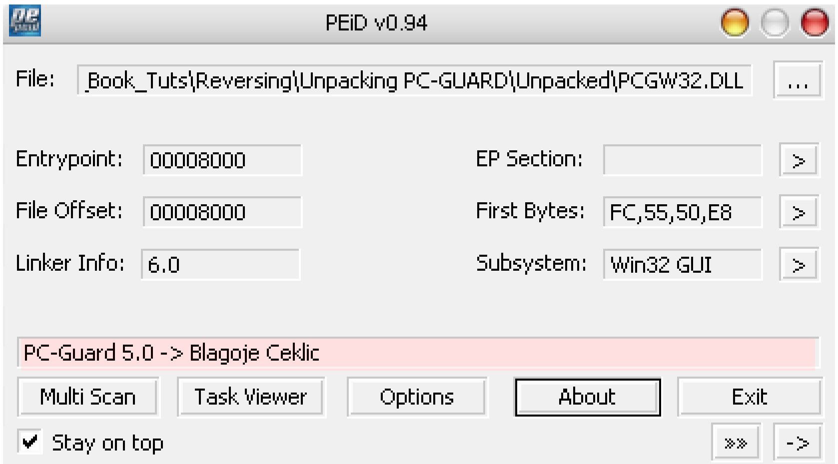


3. فك حماية ملف المكتبة الديناميكية PCGW32.DLL

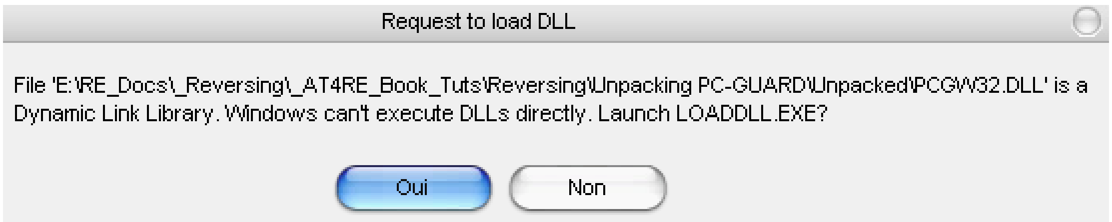
3.1 فحص ودراسة الملف

لفك حماية ملف المكتبة الديناميكية PCGW32.dll نتبع نفس الطريقة المذكورة آنفاً الخاصة باستغلال Exceptions للوصول إلى نقطة البداية الحقيقية للملف.

عملية الفحص تبين اسم برنامج الحماية :

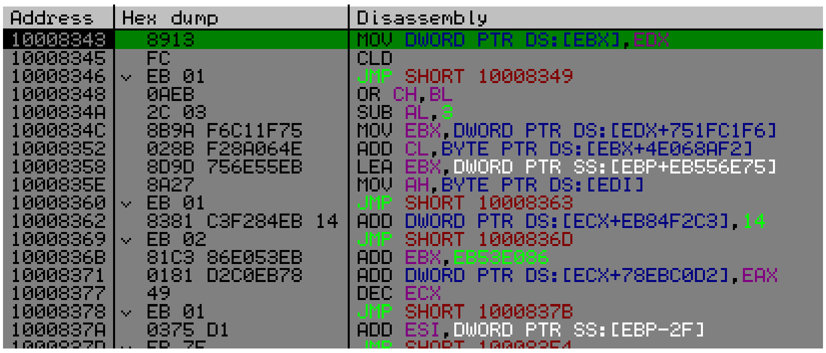


ثم :



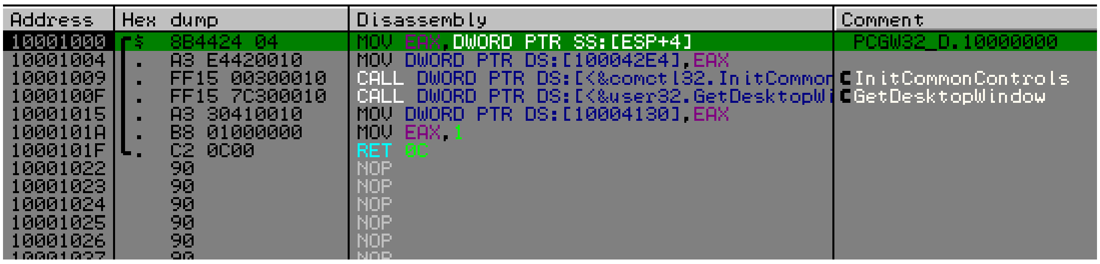
OllyDbg لا يحمّل ملفات المكتبة الديناميكية تلقائياً؛ لذا يجب علينا استعمال برنامج خارجي لتنقيح هذا النوع من الملفات، ويمكننا استعمال برنامج Loaddll.exe مع المنقح.

هذه نقطة دخول ملف Dll المحمي :

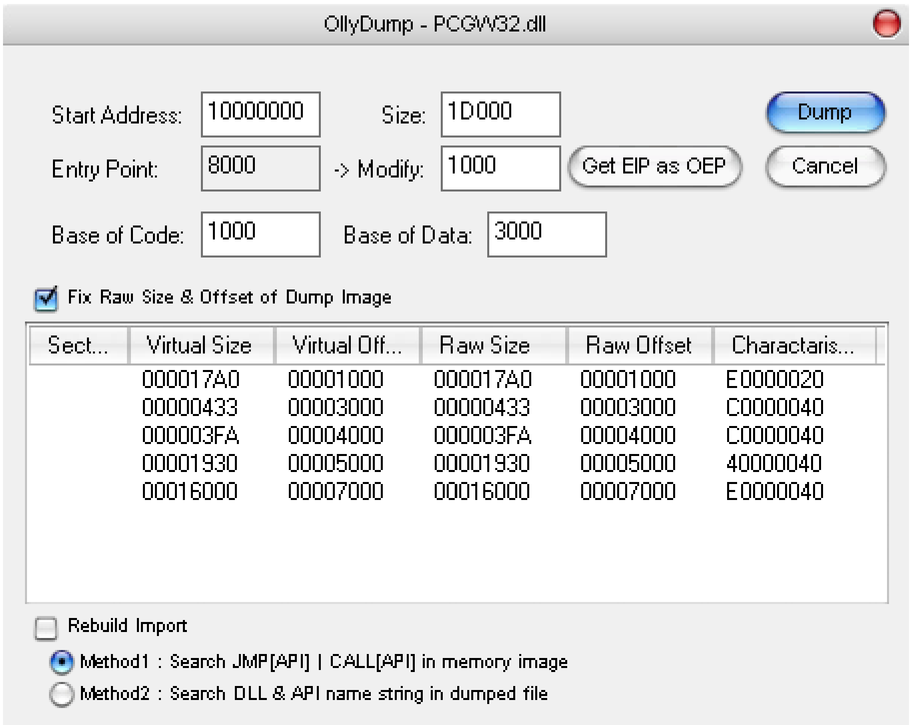


3.2 حفظ البرنامج من الذاكرة Dump

بإتباع الطريق السابقة أخي العزيز ستصل حتماً إلى نقطة البداية الحقيقية لملف Dll :



و هنا عملية حفظ الملف مباشرة من الذاكرة مع نقطة البداية الحقيقية :

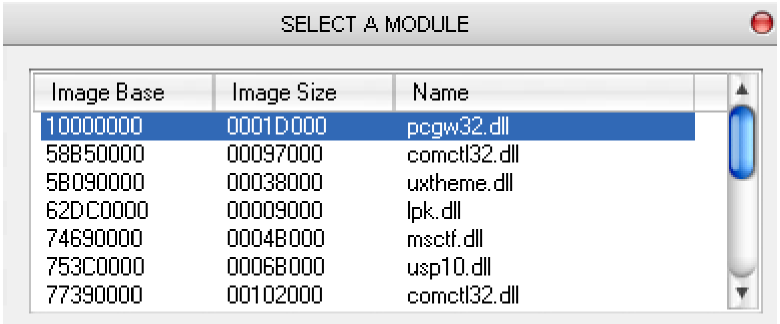


3.3 إعادة بناء Imports باستعمال برنامج Import REConstructor

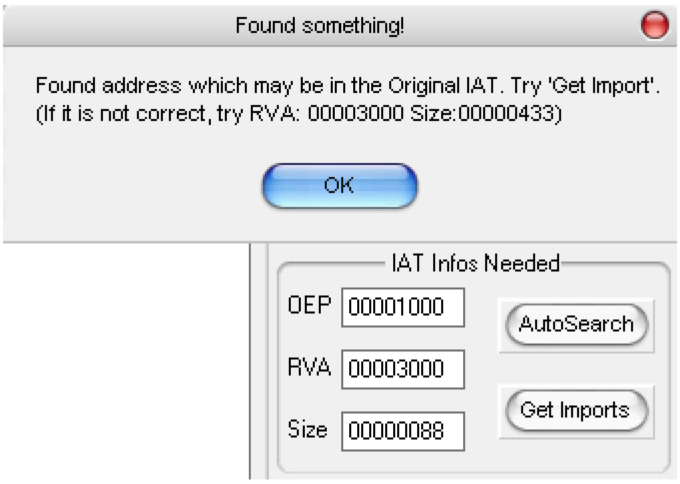


لا يمكننا تحميل ملف Dll مباشرة من الـ Process لذا سنُحمّل برنامج Loaddll.exe الذي حمل بدوره ملف Dll ثم نضغط على Pick DLL.

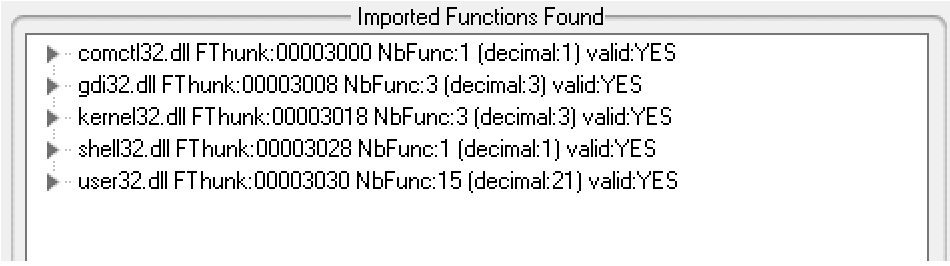
نختار الملف المراد إعادة بناء الـ Imports له في النافذة التي ظهرت لنا :



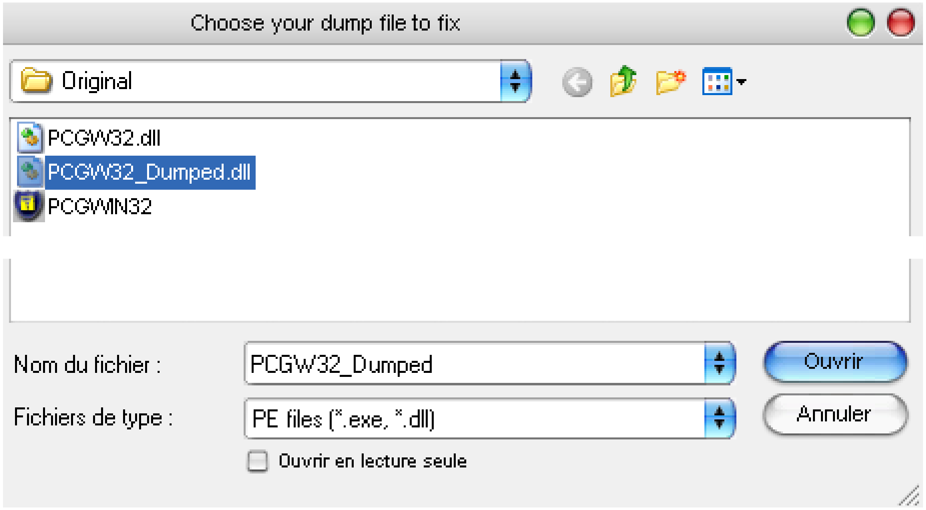
ندخل عنوان نقطة الدخول الحقيقية للملف ثم نضغط على AutoSearch ، وفي حال العثور على نتيجة إيجابية تظهر لنا رسالة نجاح العملية.



و عند الضغط على Get Imports نلاحظ أنه لا يوجد أخطاء في العناوين :



نقوم بتثبيت العناوين الصحيحة في الملف الجديد :



9. الخاتمة

الحمد لله تم هذا الدرس ونسأل الله تعالى أن يكون عظيم النفع وسهل الفهم.



الملفات التي عمل عليها هذا الدرس مرفقة ليتسنّ لك أخي الكريم التجربة مع اتباع كل المراحل والوصول إلى النتيجة المرضية. ويمكنك تحميلها من هنا.

عمل هندسة عكسية لمكونات دلفي

Delphi component RCE

1. الأهداف

الحصـــــول على معلومات التسجيـــل وعمل مولد مفاتيح Serial Fishing and Keygening.

التعديـــــل فـــي المكـــون بالـــــ Patching.

2. مقدمة

لغة البرمجة دلفي من اللغات المتطورة والسهلة الاستعمال والتخصيص، وتوفر إمكانية إضافة مكونات جديدة تجعلها قادرة على التأقلم مع متطلبات كل مبرمج مهما كان مستواه.

مثلاً يستطيع مبرمج هاوي استعمال أعقد خوارزميات التشفير الموجودة في مكتبات جاهزة دون معرفة خفاياها، وهذا ساهم في ظهور شركات متخصصة فقط في برمجة مكونات دلفي وتسويقها.

2.1 صيغة المكونات المستعملة

دلفي مزود بأداة مختصة بإضافة مكونات جديدة، وتثبيتها بصيغة :

- Pas ملف مصدري مع إمكانية التعديل وإضافة Procedures و Functions عليه.
- Dcu مع عدم إمكانية إضافات أخرى.
- Bpl مكتبة كاملة مستقلة مع عدم إمكانية إضافات أخرى أيضاً.

للمزيد من المعلومات عن مكونات دلفي وطريقة التنصيب يمكنكم زيارة موقع دلفي للعرب المتخصص في البرمجة بالدلفي على الرابط التالي :



2.2. الأدوات المستخدمة

- المنقح OllyDbg.
- محرر هكس HexWorkshop v5.
- برنامج البحث عن الـــ Search and replace – Strings
- Delphi 7 Entreprise.

من أجل توضيح هذا الدرس قمنا ببرمجة مكون دلفي، وأضفنا إليه الطرق الكلاسيكية التي تعتمد على التأكد من تشغيل دلفي لكي يعمل المكون بطريقة غير محدودة، والطرق الجديدة التي تعتمد الحصول على معلومات تسجيل دون اللجوء إلى فحص تشغيل دلفي.

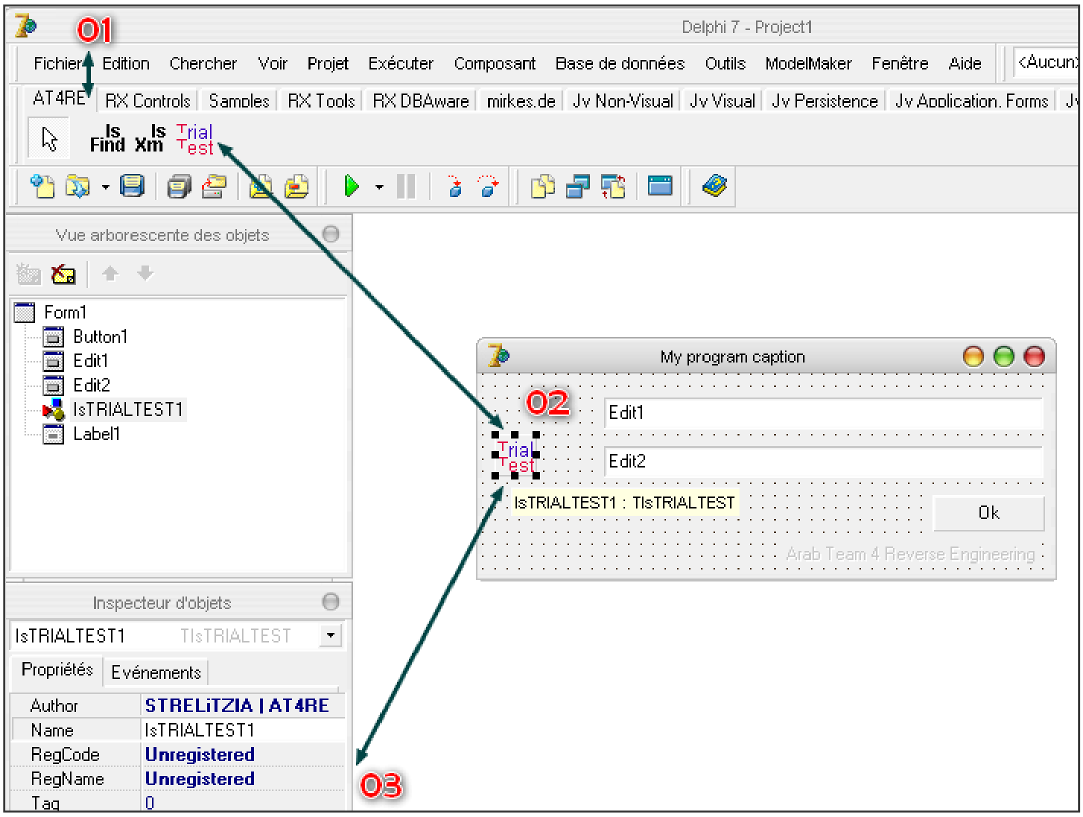
نستطيع حماية المكون بنفس الطرق المعهودة للبرامج المسقلة بدمج Junkcode أو استعمال Obfuscation، والتشفير يساعد أيضا في جعل الأجزاء الحساسة من الـــ Code غير مفهومة أثناء التتبع في المنقح.

3. الهندسة العكسية للهدف

3.1. التحليل الأولي

نقوم أولاً بتثبيت المكون IsTRIALTEST ولتسهيل العملية قمنا ببرمجة أداة تثبيت يمكنكم تحميل المكون والأداة من هنا.

نقوم بإنشاء مشروع جديد وبعد إتمام العملية نذهب للتبويب الجديد AT4RE ونضيف المكون IsTRIALTEST إلى المشروع ثم نحفظ الكل ثم نشغل المشروع، انظر الصورة :



01	التبويب AT4RE
02	إضافة المكون IsTRIALTEST
03	المعلومات الخاصة بالمكون وحقول التسجيل

نلاحظ أن البرنامج يعمل بطريقة عادية، ومظهره مطابق للمشروع دون أي تغيير أو إضافات :



نعيد تشغيل المشروع الجديد ولكن بإغلاق دلفي هذه المرة.

نلاحظ أنه أخرج رسالة يعلمنا فيها بأننا نستعمل نسخة تجريبية Trial Version :



بعد القيام بإغلاق الرسالة يظهر لنا البرنامج ولكن يوجد تغييرات في المظهر، وهناك عبارة Trial Version أضيفت إلى Caption البرنامج؟



نقوم بإغلاق المشروع وعندها سنلاحظ أنه مرة أخرى قد أخرج رسالة يعلمنا فيها بأننا نستعمل نسخة تجريبية Trial Version :



إذن فلملكون قام بالتغييرات التالية في حالة إدخال معلومات تسجيل غير صحيحة وعدم تشغيل دلفي :

01	إضافة كلمة TRIAL VERSION لعنوان البرنامج.
02	جعل واجهة البرنامج شفافة.
03	لون الخط الأسود المستعمل أصبح شفافاً أيضاً.
04	رسالة مزعجة NagScreen عند تشغيل البرنامج.
05	رسالة مزعجة NagScreen عند إغلاق البرنامج.

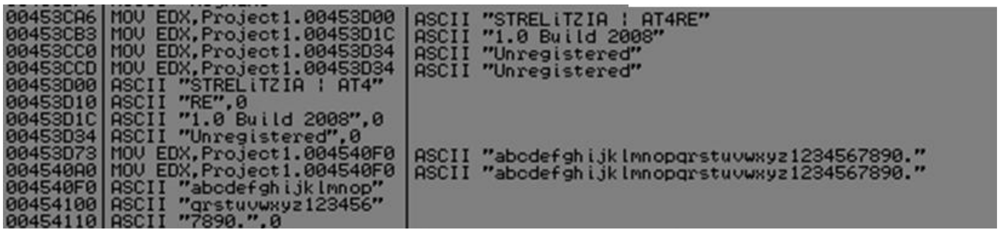
3.2 التنقيح والتتبع

نقوم بتحميل البرنامج في المنقح OllyDbg ثم نبحث عن الرسالة المزعجة إما عن طريق الكلمات المستعملة في الرسالة مثل Trial في Text strings referenced أو عن طريق استدعاءات الرسالة Windows API.

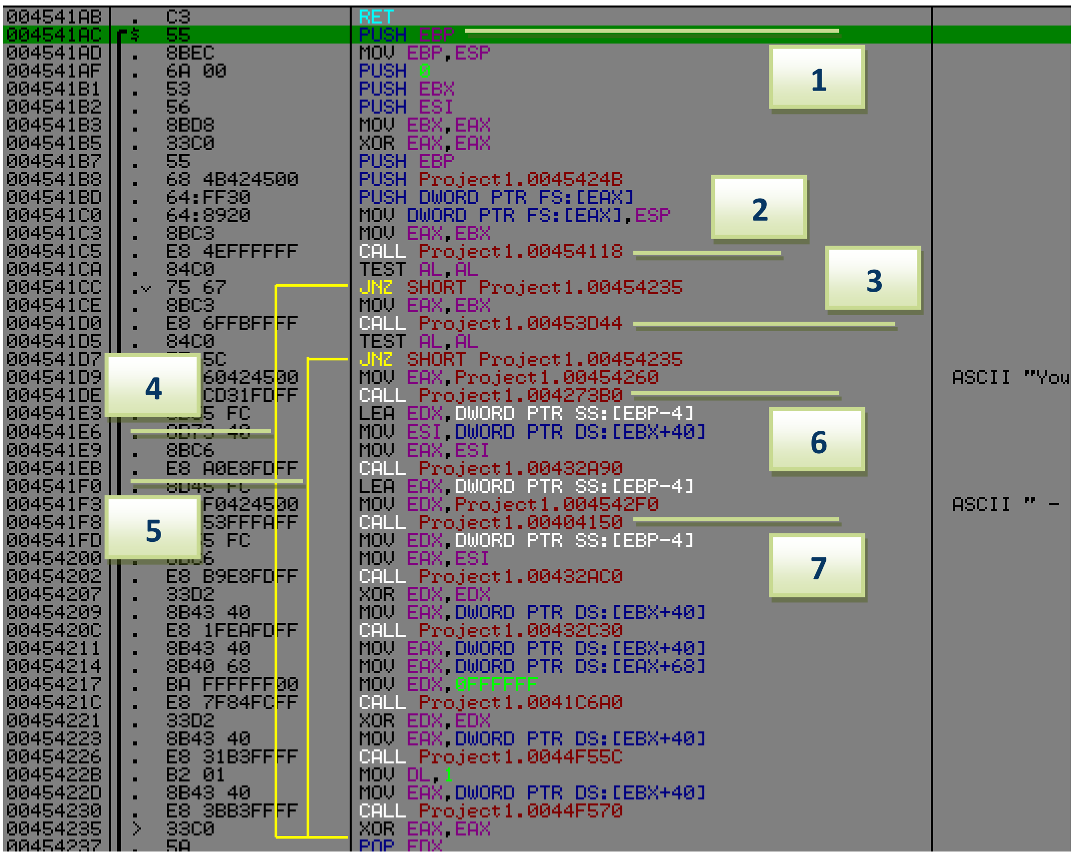
بما أنه لا يوجد تشفير للـــ Strings فالكلمات المستعملة في الرسالة تظهر بوضوح في المنقح سنقوم الآن بالبحث والتتبع.

و كما ذكر آنفا في بداية الدرس إننا نستطيع تعقيد تتبع الرسائل والكلمات التي يعتمد عليها الكثير من الـــ Reverse engineer في البحث عن عناوين الفحص بعمل Obfuscation للمناطق الحساسة.

نرى في الصورة فوق بوضوح محتويات الرسائل وأيضا كلمة TRIAL VERSION التي أضيفت إلى عنوان البرنامج.



و هنا يتبين الفحص العام التابع لـــ OnCreate واستدعاءات الفحص :



في الجدول باقي الاستدعاءات Calls – المسؤولة عن تغيير لون واجهة البرنامج وجعلها شفافة– تظهر أيضا بين القفزات المذكورة.

الرقم	ملاحظة	تعليق	تفكيك	العنوان
01	نضع نقطة توقف	البداية	PUSH EBP	004541AB
02	نضع نقطة توقف	استدعاء خوارزمية التسجيل	CALL 00454118	004541C5
03	نضع نقطة توقف	استدعاء خوارزمية التأكد من وجود دلفي في الذاكرة	CALL 00453D44	004541D0
04		القفز بعد التحقق من معلومات التسجيل الصحيحة	JNZ 00454235	004541CC
05		القفز بعد التحقق من وجود دلفي في الذاكرة	JNZ 00454235	004541D7
06		استدعاء الرسالة المزعجة	CALL 004273B0	004541DE
07		إضافة Trial Version إلى Caption البرنامج	CALL 00404150	004541F3

3.3 الحصول على معلومات التسجيل وعمل مولد مفاتيح Serial Fishing + Keygening

يبدأ استدعاء خوارزمية معلومات التسجيل كما هو موضح في الصور التالية :

بداية الاستدعاء

```
@Project1_00454118:
        push ebp

        mov ebp,esp
        add esp,-8
        push ebx
        xor edx,edx
        mov dword ptr ss:[ebp-8],edx
        mov ebx,eax
        xor eax,eax
        push ebp
        push Project1_0045419e
        push dword ptr fs:[eax]
        mov dword ptr fs:[eax],esp
        xor eax,eax
        mov dword ptr ss:[ebp-4],eax
        mov eax,dword ptr ds:[ebx+$038]
        call @Project1_00404148
        test eax,eax
        jl @Project1_00454157
        inc eax
```

```
          xor edx,edx
@Project1_00454148:
          mov ecx,dword ptr ds:[ebx+$038]
          movzx ecx,byte ptr ds:[ecx+edx-
1]
          add dword ptr ss:[ebp-4],ecx
          inc edx
          dec eax
          jnz @Project1_00454148
@Project1_00454157:
          mov eax,dword ptr ss:[ebp-4]
          mov ecx,$0256
          cdq
          idiv ecx
          imul eax,edx,$01024
          lea ecx,dword ptr ss:[ebp-8]
          mov edx,$010
          call @Project1_00407E38
          mov edx,dword ptr ss:[ebp-8]
          mov eax,dword ptr ds:[ebx+$03c]
```

معالجة اسم المستعمل تمر بعمليات حسابية بسيطة سهلة الفهم والتتبع في المنقح...

تخزين رقم التسجيل الصحيح الناتج بعد المعالجة...

تخزين رقم التسجيل الخاطئ المدخل...

خوارزمية المعالجة

```
User name = STRELiTZIA
53+54+52+45+4C+69+54+5A+49+41 = 32B
32B mod 256 = D5
D5 * 1024 = D6DF4
Format 16 hex = 00000000000D6DF4
```

عمل مولد أرقام بلغة الدلفي :

```
function GetValidSerial(Name: string): string;
var i, Serial: integer;
begin
  Serial := 0;
  for i := 0 to Length(Name) do
  begin
    Serial := Serial + Ord(Name[i]);
  end;
  Serial := Serial mod $256;
  Serial := Serial * $1024;
  Result := IntToHex(Serial, 16);
end;

procedure TKeygen.GenerateBtnClick(Sender: TObject);
begin
  SerialEdt.Text := GetValidSerial(NameEdt.Text);
end;
```

3.4. التعديل في المكون بال PATCHING

3.4.1. تحديد مكان الباتش

التعديل بالباتشينغ ليس عملية تغيير بايتات فقط دون محاولة فهم ما يدور في البرنامج المستهدف، في كثير من الحالات تغيير بايت واحد فقط يكفي أن يجعل البرنامج مسجلاً بشكلٍ كامل دون تغيير عدة قفزات هنا وهناك.

بداية الاستدعاء

```
@Project1_004541ac:
        push ebp
        mov ebp,esp
        push 0
        push ebx
        push esi
        mov ebx,eax
        xor eax,eax
        push ebp
        push Project1_0045424b
        push dword ptr fs:[eax]
        mov dword ptr fs:[eax],esp
        mov eax,ebx
        call @Project1_00454118      <-- استدعاء فحص معلومات التسجيل
        test al,al
        jnz @Project1_00454235       <-- قفزة شرطية إلى بر الآمان في حالة al = 1
        mov eax,ebx
                                     <-- استدعاء فحص تشغيل دلفي في الذاكرة
        call @Project1_00453D44
        test al,al                   <-- قفزة شرطية إلى بر الآمان في حالة al = 1
        jnz @Project1_00454235
        mov eax,Project1_00454260
@Project1_004273B0
        lea edx,dword ptr ss:[ebp-4]
        mov esi,dword ptr ds:[ebx+$040]
```

نلاحظ أن القفزة الأولى الخاصة بفحص معلومات التسجيل تقفز أيضاً فوق استدعاء التحقق من وجود دلفي في الذاكرة مما يجعلنا نضرب عصفورين بحجرٍ واحد.

سنقوم بتغيير القفزة الشرطية إلى قفزة غير شرطية :

```
        call @Project1_00454118
        test al,al
        jmp @Project1_00454235       <-- JNZ تصبح JMP
        mov eax,ebx
        call @Project1_00453D44
        test al,al
        jnz @Project1_00454235
```

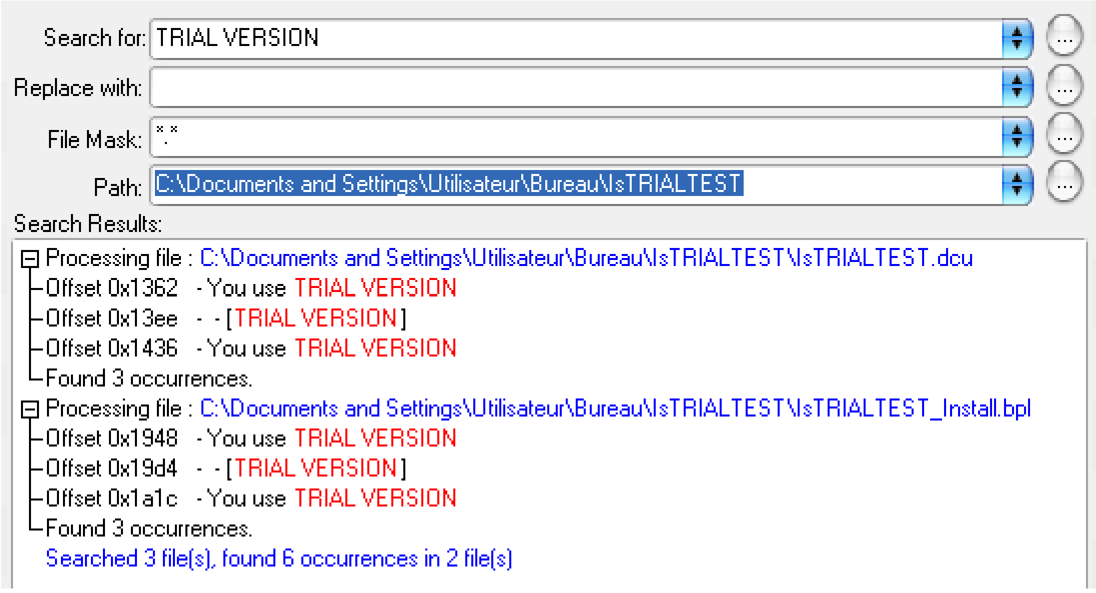
بعد حفظ التغييرات نجرب تشغيل البرنامج من جديد...

رائع!! البرنامج يعمل بكفاءة ولكن هل يجب علينا في كل مرة عمل باتشينغ لكل مشروع جديد؟

الجواب: لا طبعاً، والحل: هو التعديل المباشر في المكون عن طريق المحرر HexWorkshop v5.

3.4.2 طريقة التعديل

في معظم الحالات تكون الملفات المرفقة في Delphi package كثيرة جداً، ويكون البحث في كل ملف عن كلمة Trial متعب جداً؛ لذا استعمال الأداة Search and replace يكون مفيداً جداً.



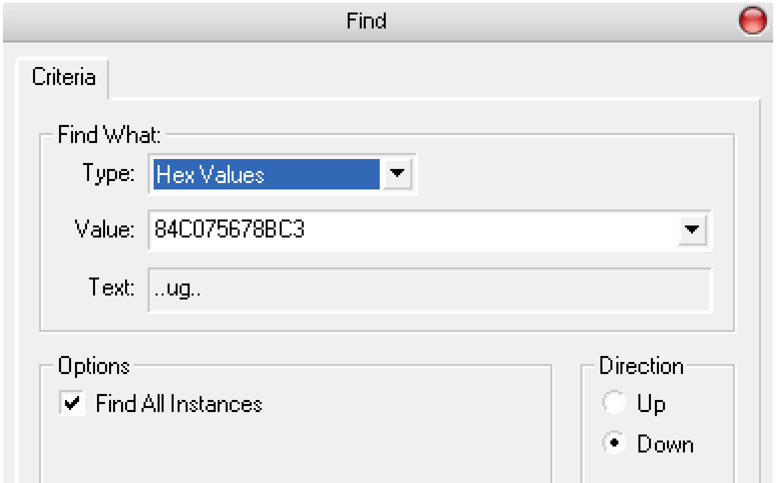
جيد، يمكننا الآن فتح الملفات المسؤولة عن الرسائل والتغييرات الخاصة بـ Trial Version ولكن قبل ذلك نأخذ القيم المبينة في الصورة التالية :

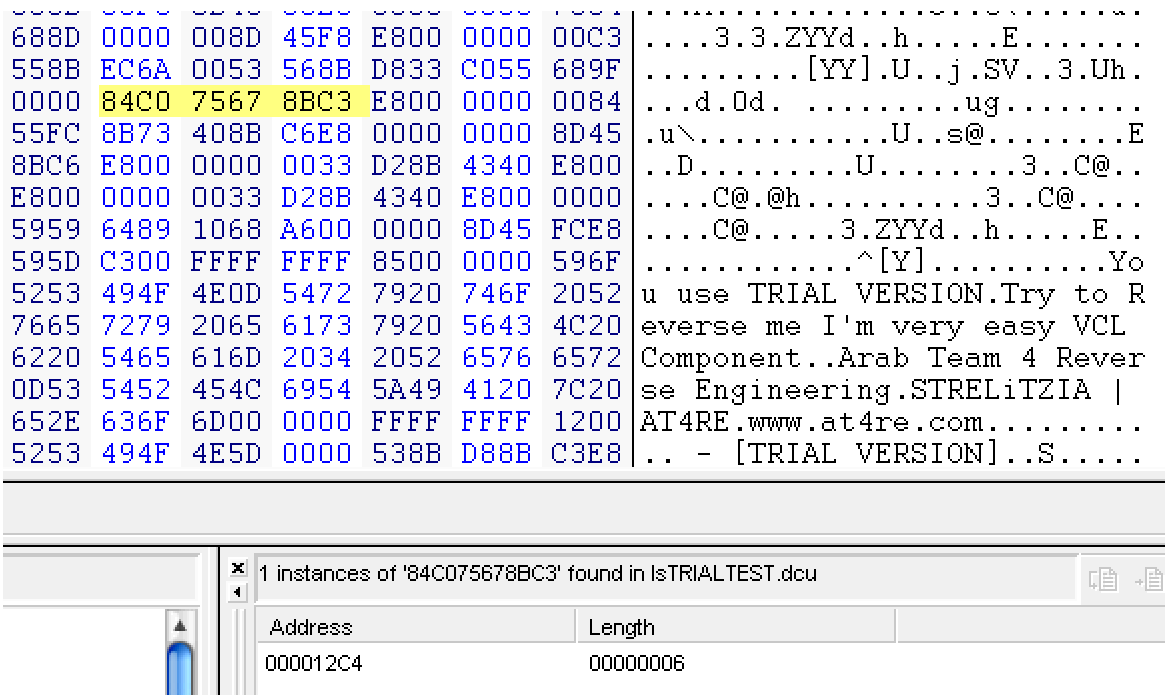
```
004541C5   E8 4EFFFFFF     CALL Project1.00454118
004541CA   84C0            TEST AL,AL
004541CC   75 67           JNZ SHORT Project1.00454235
004541CE   8BC3            MOV EAX,EBX
004541D0   E8 6FFBFFFF     CALL Project1.00453D44
004541D5   84C0            TEST AL,AL
```

نأخذ هذا الجزء من الكود :

84C075678BC3

تسهيل البحث في محرر Hex





تعديل ملف IsTRIALTEST.dcu

قبل : 000012B4 |0000 0064 FF30 6489 208B C3E8 0000 0000 84C0 7567 8BC3 E

بعد : 000012B4 |0000 0064 FF30 6489 208B C3E8 0000 0000 84C0 EB67 8BC3 E

تعديل ملف IsTRIALTEST_Install.bpl

قبل : 0000189C |0064 FF30 6489 208B C3E8 4EFF FFFF 84C0 7567 8BC3 E8

بعد : 0000189C 0064 FF30 6489 208B C3E8 4EFF FFFF 84C0 EB67 8BC3 E86

بعد الانتهاء من التعديل نقوم بإعادة تثبيت مكون دلفي إذا اقتضت الضرورة، وخصوصاً في حالات عمل compilation. وبالنسبة لتمريننا فلا يقتضي الأمر ذلك.

برمجة إضافة لـ OllyDbg

Developping an OllyDbg Plugin

- OllyDbg.
- [Plugin Development Kit 1.10](#).
- A c/c++ compiler/linker (سنستعمل C/C++ IDE سنستخدم vs2008).

(التحميل من مرفقات الكتاب).

I. مقدمة

إضافات OllyDbg عبارة عن ملفات DLL عادية، لكن يشترط أن تصدّر على الأقل دالتين من أصل 14 دالة معرفة في ملف plugin.h .

عندما يبدأ OllyDbg عمله فإنه يقوم بتحميل جميع الملفات ذات الامتداد "DLL" في مجلد الإضافات واحدةً تلو الأخرى، ويفحص إذا ما كانت تصدر هذه الدوال.

```
_ODBG_Plugindata
_ODBG_Pluginint
```

إذا كان ملف "DLL" يصدر هاتين الدالتين فإن OllyDbg – مبدئياً– يعتبر أن الملف عبارة عن إضافة صالحة.

إذن: الآن لدينا معلومات كافية بشكل نظري لإنشاء إضافة لـــ OllyDbg، وسنقوم بالآتي :

1. إنشاء مشروع جديد من نوع "Dynamic link library".
2. إنشاء الدالتين _ODBG_Plugindata و_ODBG_Plugininit وتصديرهما.

2. إنشاء مشروع جديد من نوع "Dynamic link library"

- شغّل الفيجوال ستوديو.
- من قائمة File اختر New >> Project.
- من قائمة نوع المشروع اختر Visual C++ >> Win32.
- من القائمة المقابلة اختر Win32 Project.
- أدخل اسم المشروع الجديد بدلاً من "<Enter_name>" وليكن "MyOlly" ثم اضغط موافق، سيظهر لك المعالج اختر "DLL" كنوع المشروع و"Empty Project"\"مشروع فارغ" ثم اضغط موافق.
- الآن سنقوم بإضافة ملف كود جديد\فارغ للمشروع بأي اسم وبامتداد cpp وحان وقت كتابة الكود :

```
BOOL APIENTRY DllMain( HINSTANCE hModule, DWORD  ul_reason_for_call, LPVOID
lpReserved)  {
      switch (ul_reason_for_call) {
            case DLL_PROCESS_ATTACH:
                  g_hInst=hModule;
      }
      return TRUE;
}
```

هذه الدالة عبارة عن المدخل \"Entry point" وهي ثابتة في معظم ملفات "DLL"

قمنا بتعريف متغير عام "g_hInst" لحفظ قيمة HINSTACE الخاصة بالإضافة

الآن انتهينا من الجزء الأول وسننتقل إلى الخطوة التالية والتي سنقوم فيها بكتابة الدالتين المذكورتين سابقاً.

3. إنشاء الدالتين _ODBG_Plugindata و_ODBG_Plugininit وتصديرهما

الآن بما أننا لا نعرف ما الذي توقعه OllyDbg من هذه الدوال عندما يقوم بتحميل الإضافة سنقوم بالرجوع إلى ملف المساعدة المرفق \plugin.hlp ، افتح الملف ثم انتقل إلى قسم "Plugin callback functions"، وسترى 14 تعريفاً للدوال التي يمكن للإضافة تصديرها. هذه الدوال هي وسيلة التخاطب بين الإضافة و OllyDbg، وما يهمنا منها حاليا الدالتين ODBG_Plugindata وODBG_Plugininit.

لنلقي نظرة على الدالة الأولى :

```
int ODBG_Plugindata(char *shortname) ;
```

هذه الدالة يجب أن تكون موجودة في جميع إضافات OllyDbg، ووظيفتها باختصار نسخ اسم الإضافة للمتغير shortname وإرجاع رقم نسخة OllyDbg التي نبرمج الإضافة لها، وفي جميع الحالات ستكون هذه القيمة مساوية للثابت PLUGIN_VERSION.

> اسم الإضافة يجب أن لا يزيد عن 31 حرفً وأن لا يحتوي على أي رموز خاصة.

الآن لنقم بتطبيق هذه الدالة :

```
int ODBG_Plugindata(char *shortname) {
      lstrcpy(shortname, "MyOlly");
      return PLUGIN_VERSION;
}
```

> ما الذي فعلناه ؟
>
> قمنا باستخدام الدالة lstrcpy لنسخ اسم الإضافة التي نقوم بإنشائها إلى المتغير shortname ثم قمنا بإرجاع القيمة PLUGIN_VERSION والتي تكافئ 110 رقم إصدار OllyDbg الذي نبرمج الإضافة له.

الآن لننتقل إلى الدالة الثانية التي يجب أن نقوم بتطبيقها في الإضافة ODBG_Plugininit.

هذا هو التعريف الذي يعطينا إياه ملف المساعدة للدالة :

int **ODBG_Plugininit** (	
int OllyDbgversion,	رقم إصدار OllyDbg.
HWND hw,	مقبض النافذة الرئيسية في OllyDbg.
ulong *features	قيمة محجوزة لإصدارات قادمة من OllyDbg وبالتالي لا تهمنا.
);	
Return value	الدالة يجب أن تعيد القيمة 0 إذا أردنا أن يقوم OllyDbg بتحميل الإضافة بنجاح أو 1- لكي يقوم OllyDbg بتجاهل الإضافة.

الآن لنقم بتطبيق هذه الدالة :

```
int ODBG_Plugininit(int OllyDbgversion,HWND hw,ulong *features) {
	if (OllyDbgversion<PLUGIN_VERSION) {
		return -1;
	}

	g_hOlly=hw;
	return 0;
}
```

في السطر الأول نقوم بمقارنة الثابت PLUGIN_VERSION والذي يساوي 110 مع المتغير OllyDbgversion الذي يمرره OllyDbg للإضافة، إذا كانت قيمة OllyDbgversion أصغر من PLUGIN_VERSION فإننا سنقوم بإرجاع 1- لكي يتم تجاهل الإضافة.

متى يمكن أن تكون قيمة OllyDbgversion أصغر من PLUGIN_VERSION ؟

كما تعرف فإن النسخة الأخيرة من OllyDbg هي 1.10 وهي النسخة التي نقوم ببرمجة هذه الإضافة لها.

تصور لو أن أحداً ما لا يزال يستخدم نسخة قديمة مثلا 1.08، في هذه الحالة عندما تقوم هذه النسخة القديمة من OllyDbg بتحميل الإضافة سيتم تنفيذ الدالة ODBG_Plugininit ومن ثم التحقق من الشرط OllyDbgversion<PLUGIN_VERSION.

بما أننا برمجنا الإضافة لنسخة OllyDbg 1.10 فستكون قيمة PLUGIN_VERSION تساوي 110، لكن الآن يتم تحميل الإضافة في نسخة قديمة من OllyDbg لذلك ستكون قيمة OllyDbgversion تساوي رقم هذه النسخة القديمة في الحالة التي افترضناها 108.

سؤال آخر: ماذا لو قررنا تجاهل هذا الأمر، وفضلنا المتابعة في تحميل الإضافة، بمعنى آخر أرجعنا القيمة 0؟

كثير من الأشياء غير المتوقعة يمكن أن تحدث ابتداءً من أن الإضافة لا تعمل بشكل صحيح وانتهاءً بتحطم OllyDbg .

لننتقل إلى السطر التالي :

```
g_hOlly=hw
```

هنا عرفنا متغير عام "g_hOlly" لكي نحفظ مقبض نافذة OllyDbg فيه، والتي سوف تحتاجها لاحقاً بالتأكيد إذا قررت استخدام بعض دوال الـــ API.

في السطر الذي يليه قمنا بإرجاع القيمة 0 من الدالة لكي يستمر OllyDbg في تحميل الإضافة.

الآن لدينا إضافة يقبلها OllyDbg لكن تبقي بغير فائدة, لنقم بتطويرها قليلا..

ما الذي نريد عمله الآن ؟

إضافة عنصر إلى قائمة OllyDbg عند تفعيله يقوم بتتبع تعليمة الأسمبلي المختارة في نافذة الدمب.

لكي نقوم بإضافة عناصر جديدة إلى قوائم OllyDbg المختلفة يلزمنا تطبيق دالتين أخرتين من الدوال الــ 14 التي رأيناها في ملف "plugin.hlp" :

- أول دالة هي ODBG_Pluginmenu وهي الدالة التي تسمح لنا بإضافة عناصر جديدة للقوائم.
- الدالة الثانية هي ODBG_Pluginaction التي يستدعيها OllyDbg عند تفعيل أحد عناصر القوائم الخاصة بالإضافة.

لنرى الدالة الأولى :

int ODBG_Pluginmenu (	
int origin,	معرف النافذة الحاضنة للقائمة التي تم تفعيل أحد عناصرها.
char data[4096],	مخزن يتم فيه نسخ أسماء العناصر التي نريد إضافتها. لاحظ أن أقصى حجم مسموح به هو 4096 بايت.
void *item	عنوان للعنصر المختار في أحد نوافذ OllyDbg.
);	يتوقع OllyDbg منا أن نعيد القيمة 1 إذا أضفنا عناصر جديدة لإحدى القوائم أو القيمة صفر في حال لم نقم بإضافة أي عناصر جديدة.

لنقم بتطبيق الدالة :

```
int ODBG_Pluginmenu(int origin,char data[4096],void *item) {
      switch (origin) {
            case PM_DISASM:
                  {
                        char items[1024] = { '\0' };
                        wsprintf(items,"0 Follow selection");
                        lstrcpy(data, items);
                        return 1;
                  }
            default: break;
      }
      return 0;
}
```

لنلقي نظرة على الكود :

- قمنا بتطبيق جملة الاختيار switch على المتغير origin والذي يحتوي على معرف لأحد نوافذ OllyDbg المختلفة.
- ما يهمنا من هذه المعرفات الآن PM_DISASM التي تشير إلى نافذة OllyDbg التي تعرض تعليمات الأسمبلي.

لنري الكود في الأعلى حيث قيمة origin تساوي PM_DISASM.

قمنا بإنشاء مخزن كبير بشكل كافي "1024 بايت" لتخزين أسماء العناصر الجديدة التي سيتم إضافتها إلى القائمة الخاصة بنافذة الأسمبلي، ثم قمنا بنسخ السلسلة النصية إلى المخزن بواسطة lstrcpy ، وأعدنا القيمة 1 لكي نخبر OllyDbg أننا أضفنا عناصر للقائمة ونتوقع أن تظهر في القائمة الخاصة بنافذة الأسمبلي.

الان لنتعرف على النسق المستخدم في بناء عناصر القائمة.

في مثالنا السابق قمنا بنسخ "Follow selection 0" إلى المخزن data ما يعنيه هذا أننا نريد من OllyDbg أن يضيف عنصر جديد إلى القائمة باسم "Follow selection" ويعطي هذا العنصر معرف قيمته 0.

هذا هو النمط العام :

اسم القائمة الخاصة بـــ الإضافة الذي نقوم بإنشائها (اختياري).

} هذا الرمز يستخدم كدلالة على بداية القائمة الفرعية (اختياري).

- معرف العنصر الأول.
- اسم العنصر الفرعي الأول.

| هذا الرمز يستخدم لإضافة فاصل (اختياري).

- معرف العنصر الثاني (اختياري).
- اسم العنصر الفرعي الثاني (اختياري).

{ هذا الرمز يستخدم كدلالة على نهاية القائمة الفرعية (اختياري).

* ارجع لملف plugin.hlp لمزيد من التفاصيل..

الآن لنرى الدالة التالية التي يجب تطبيقها :

void ODBG_Pluginaction (	
int origin,	معرف النافذة الحاضنة للقائمة التي تم تفعيل أحد عناصرها.
int action,	معرف العنصر الذي تم تفعيله من القائمة (0 في حالتنا, ارجع للدالة السابقة).

عنوان للعنصر المختار في أحد نوافذ OllyDbg.	void *item
	);

```
void ODBG_Pluginaction(int origin,int action,void *item) {
      if(origin==PM_DISASM) {
            switch (action) {
                  case 0:
                  {
                        t_dump *pd= (t_dump *)item;
                        if (pd!=NULL && pd->size!=0) {
                              Setcpu(0,0,pd-
>sel0,0,CPU_DUMPHIST|CPU_DUMPFIRST|CPU_DUMPFOCUS);
                        }
                  }
                  break;
                  default: break;
            }
      }
}
```

لنرى الكود :

- قمنا باختبار ما إذا كانت قيمة origin تساوي PM_DISASM إذا كانت النتيجة إيجابية فإننا سننتقل إلى الخطوة الأخرى وهي اختبار قيمة action مع 0.
- بمعنى آخر هل قام مستخدم الإضافة بتفعيل العنصر "Follow selection" الذي يحمل قيمة معرف 0 ؟

إذا كانت النتيجة إيجابية فسنقوم بتنفيذ دالة Setcpu وهي واحدة من بين عشرات الدوال التي يوفرها OllyDbg لمبرمجي الإضافات.

ما تقوم به هذه الدالة هو إظهار عنوان معين تختاره أنت في أحد نوافذ OllyDbg الثلاثة Cpu,dump,stack.

* ارجع للملف plugin.hlp لمزيد من التفاصيل

الآن انتهينا من برمجة الإضافة وبقي علينا بعض الأشياء البسيطة :

1. تصدير الدوال الأربع لــ OllyDbg : سنقوم بعمل ذلك بإضافة بعض الكلمات المحجوزة إلى ترويسة هذه الدوال:

- _export : لتصدير الدالة.
- cdecl : لمنع الكمبايلر من إعادة تسمية هذه الدوال.

فلو أخذنا دالة ODBG_Pluginaction كمثال فسنقوم بالتعديل على الترويسة لتصبح :

```
void  _export cdecl ODBG_Pluginaction(int origin,int action,void *item)
```

بدلا من

```
void ODBG_Pluginaction(int origin,int action,void *item)
```

2. إضافة الملفين plugin.h وOLLYDBG.LIB إلى المشروع.

- قم بنسخ هذين الملفين إلى أي مشروع جديد تقوم بإنشائه وقم بسحبهما إلى داخل المشروع "Darg and Drop".

3. قم بتفعيل خيار الكمبايلر J/ .

- تستطيع عمل هذا في الفيجوال ستوديو باتباع الخطوات التالية : Option > Configuration Properties > C/C++ > Language > Defualt Char Unsigned > Yes

بقي أن نشير إلى أن هذه المقال ما هو إلا بداية لك إذا أردت أن تبرمج إضافات لـــ OllyDbg.

يجب أن تجيد إحدى لغات البرمجة وتستعمل ملف plugin.hlp للإجابة عن أسئلتك المستقبلية المتعلقة بهذا الموضوع.

الباب الثاني: التعامل مع الحسابات المتقدمة

كسر برنامج مشفر ب ASPROTECT SKE بطرق مختلفة

ASPROTECT SKE UNPACKING/CRACKING

(several methods ware used)

ASPROTECT SKE
FOR WINDOWS 95/98/ME/NT4/2000/XP/2003/VISTA

اسم البرنامج الضحية : Atomic RAR Password Recovery v1.20.

وظيفته : كسر حماية ملفات rar المحمية بكلمة مرور.

الحماية : ASProtect 2.1x SKE -> Alexey Solodovnikov.

موقع الضحية هو: http://www.apasscracker.com أو اضغط على الصورة لتحميل النسخة التي تم تطبيق الدرس عليها مع مرفقاته.

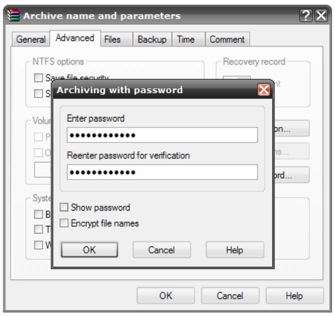
- OllyDbg v1.10.
  - HideOD+OdbgScript v1.65.
  - Aspr2.XX_unpacker_v1.14aE.osc. (في المرفقات)
- PEiD v0.95.
- Import REConstructor F v1.6.
- aspr crc locator.
- DUP v2.19.
- Stripper v2.13 beta9 or above.
- ASProtect SKE.
- Inline Patchinger v0.2.

و لنبدأ على بركة الله...

I. أولا : تحليل البرنامج

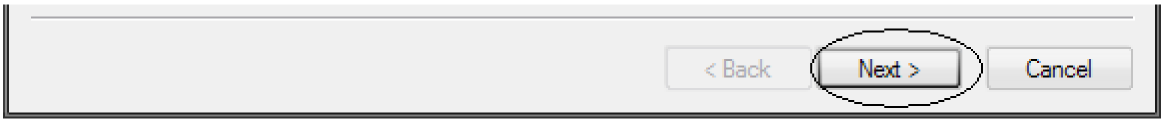
هيا بنا نتفحص عيوب النسخة الغير مسجلة.

نحتاج إلى ثلاثة ملفات rar تم حمايتها بكلمة مرور:

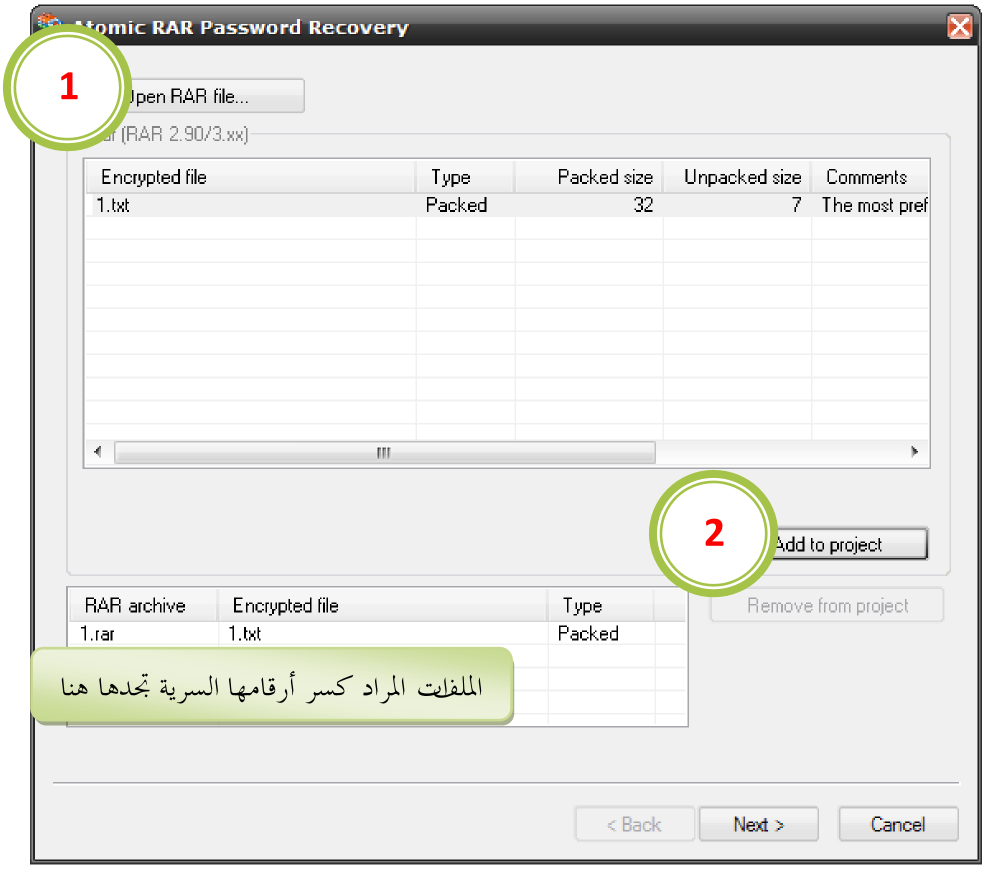




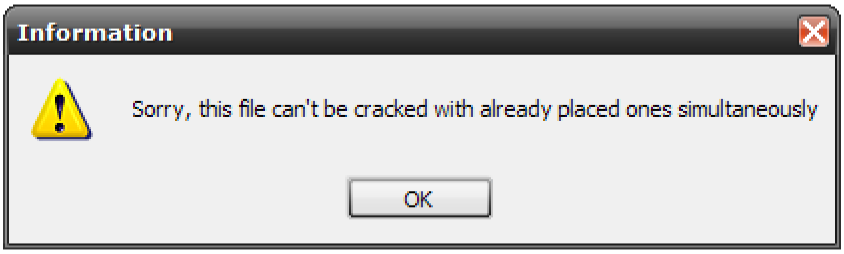
بعدها قم بفتح البرنامج الرئيسي AtomicRarPasswordRecovery.exe وانقر على Next :



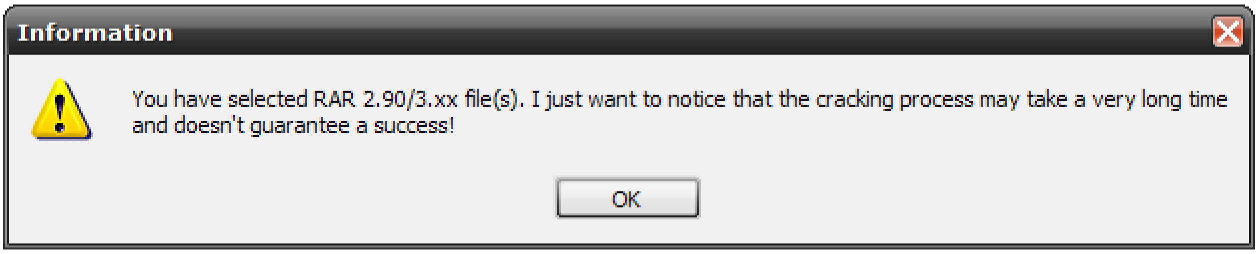
ثم انقر على Open RAR File واختر ملف rar المراد كسره ، ثم انقر على Add to project :



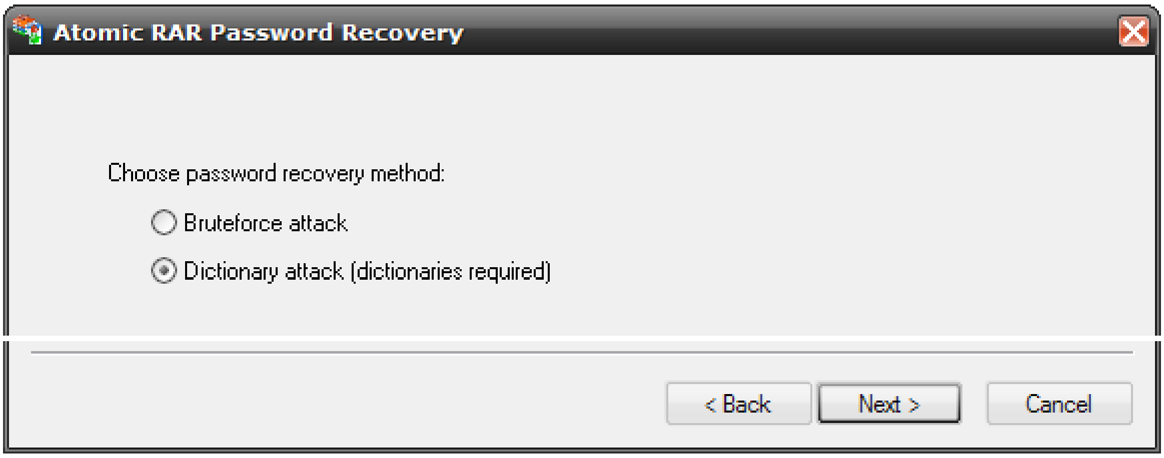
والآن عند إضافة الملف الثاني – بنفس الطريقة – تظهر هذه الرسالة المزعجة :



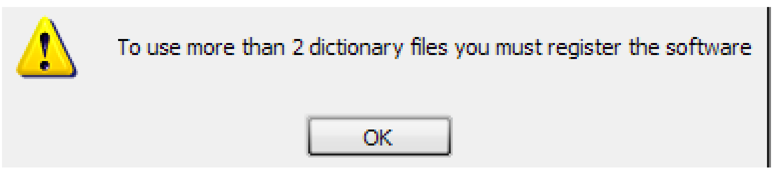
لا بأس سنزيلها فيما بعد، انقر على Next وعندها ستظهر هذه الرسالة – لا عليك منها – انقر على OK :



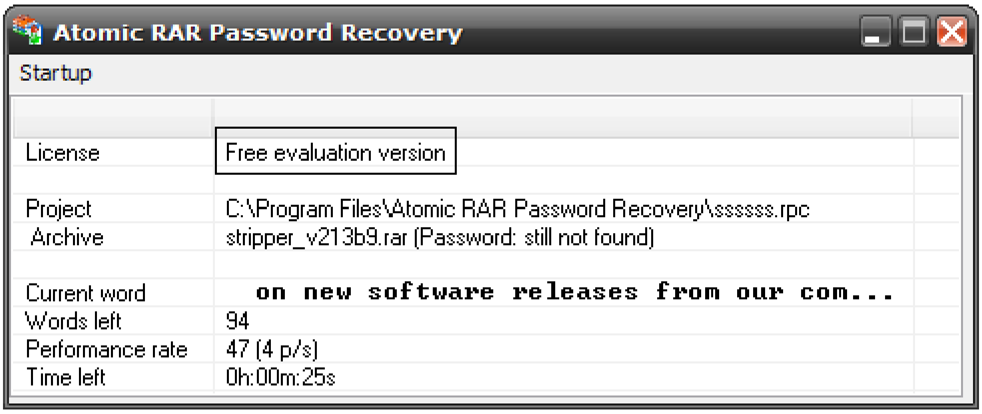
ثم مجدداً انقر على NEXT :



بعدها قم بالنقر على add وأضف 3 ملفات txt بهم جمل أو كلمات أو أي شيء وانقر على Next لتشاهد هذه الرسالة المزعجة أيضاً :

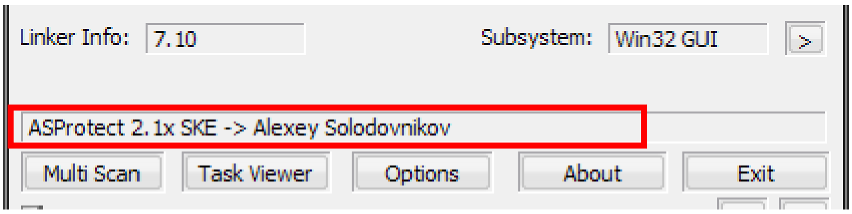


احذف ملفاً واحداً من ملفات txt وانقر على Next ثم احفظ اسم المشروع في أي مكان وانقر على Next ، بعدها سنصل إلى هنا :



تقريباً هذه أبرز الأمور المزعجة في البرنامج ، هنالك أمور أخرى ستتعرف عليها أكثر عند تنقيح البرنامج.

والآن اخرج من البرنامج وافحصه باستخدام PEiD v0.95:



إنه مشفر بـــ ASProtect 2.1x SKE ، وهنالك العديد من الطرق لفك هذه الحماية و كسر البرنامج :

1) فك البرنامج باستخدام Stripper v2.13 beta 9 : وهذا البرنامج يقوم بفك البرامج المشفرة بـــ ASProtect 2.1x SKE ، لكنه لا يعمل على الفيستا.

2) فك البرنامج باستخدام OdbgScript v1.65 : في الآونة الأخيرة اشتهرت هذه الأداة والتي يتم إضافتها إلى المنقح OllyDbg وهي تختصر الطريق لفك الكثير من البرامج، ومع ASProtect بإصداراته المتنوعة يشتهر سكريبت Aspr2.XX_unpacker_v1.14aE.osc من برمجة أحد الصينيين.

3) عمل Loader : وهذه ستتطلب منك إزالة CRC check وبعض الرسائل المزعجة، وهنالك بعض البرامج المتخصصة في عمل Loaders للحمايات المشهورة، وأحد هذه البرامج هو Advanced Loader Generator ، وكذلك هنالك سورس كود c++ من برمجه فريق ARTeam ويختص فقط بعمل Loader لـ asprotect.

4) عمل Inline Patching : وهذه الطريقة تقوم على أساس إيجاد مساحات خالية في البرنامج، بحيث يتم وضع البايتات المراد تعديلها في البرنامج في تلك المساحات، وذلك من خلال method معين. وهنالك برامج خاصة لأداء هذه المهمة أو تسهيلها مثل ASProtect SKE Inline v0.2.

5) الفك اليدوي MUP : ويتم على أساس إيجاد OEP وعمل Dump ثم تصحيح Imports يدوياً، وقد يتخلل ذلك عدة عمليات قد تكون إيجاد البايتات المسروقة أو لخبطة في الـ Imports ، ويعتمد ذلك على الضحية، لكننا لا ننصح بهذا الخيار إلا في الحالات الحرجة، هنالك مثل يقول : "آخر الدواء الكي".

1.1. الخطوة الأولى : فك البرنامج الضحية

1.1.1. الطريقة الأولى : استخدام Stripper v2.13 beta 9

افتح برنامج Stripper v2.13 beta 9 ثم انقر على Open

واختر الملف البرنامج الضحية AtomicRarPasswordRecovery 

ثم انقر على: 

ستظهر لك نافذة Imports المصححة، منها انقر على Save : 

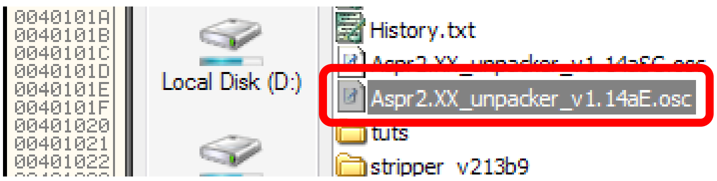
وأخيراً ستجد الملف المفكوك في نفس مجلد البرنامج باسم AtomicRarPasswordRecovery.exe_ .

ماذا فعل Stripper لفك البرنامج ؟

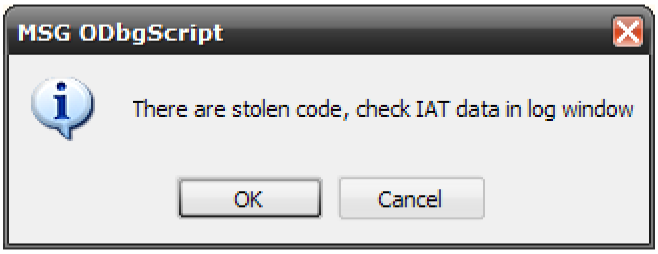
لقد قام بعمل بحث للـ OEP ثم dump victim ثم fixing imports ثم Fixing Dump File ، وهذا الحال يتم مع أي برنامج مشفر بطريقة عادية ويراد فكه.

1.1.2. الطريقة الثانية : استخدام OdbgScript v1.65

حمل البرنامج الضحية في OllyDbg ، ثم من القائمة اذهب إلى Plugins >>>OdbgScript ثم افتح سكريبت الفك Aspr2.XX_unpacker_v1.14aE.osc

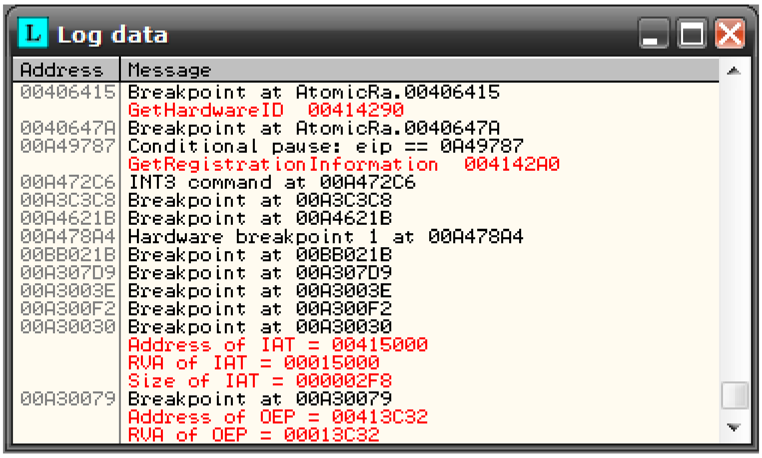


سيقوم السكريبت بعمله، وبعد قليل ستظهر هذه الرسالة :



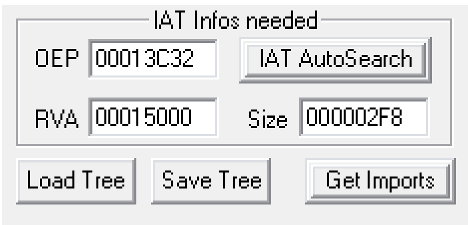
حسناً، إنه يقول: إن البرنامج الضحية لا يحتوي على خاصية البايتات المسروقة في خيارات التشفير، وهذا من حسن الحظ..

انقر على OK ثم اضغط على Alt+L :



نعم هذه العناوين الخاصة بالـ OEP والـ Imports المعدلة.

الآن افتح برنامج ImpRec وضع RVA of OEP وRVA of IAT وSize of IAT في أماكنها المخصصة وانقر على Get Imports



وأخيراً اضغط على Fix Dump واختر ملف de_AtomicRarPasswordRecovery.exe وبعد قليل ستجد الملف المفكوك باسم de_AtomicRarPasswordRecovery_.exe



1.2 الخطوة الثانية : تنقيح البرنامج

افتح البرنامج المفكوك de_AtomicRarPasswordRecovery_.exe أو AtomicRarPasswordRecovery_.exe وابدأ البحث عن الــ nags التي شاهدناها والتي لم نشاهدها.

كل تلك الأشياء المزعجة يمكن إزالتها وذلك عن طريق عكس القفزة الموجودة أعلى كل واحدة منها

ASCII "To find passwords witch have a dynamic part longer than 3 characters in …"

ASCII "To crack more than 2 files simultaneously you must register software"

ASCII "To use more than 2 dictionary files …"

ASCII "Sorry, this file can't be cracked with already placed ones simultaneously"

ASCII "Fully functional version"

```
74 22            JE SHORT de_Atomi.00401881
2BC1             SUB EAX,ECX
83F8 03          CMP EAX,3
77 1B            JA SHORT de_Atomi.00401881
68 C0534100      PUSH de_Atomi.004153C0          ASCII "To find passwords which have a
```

```
74 47            JE SHORT de_Atomi.00403010
66:833D 705C4:   CMP WORD PTR DS:[425C70],2
77 0A            JA SHORT de_Atomi.00402FDD
68 28574100      PUSH de_Atomi.00415728          ASCII "To crack more than 2 files simu
```

```
75 14            JNZ SHORT de_Atomi.00402FFA
66:833D B85D4:   CMP WORD PTR DS:[425DB8],2
77 0A            JA SHORT de_Atomi.00402FFA
68 E0564100      PUSH de_Atomi.004156E0          ASCII "To use more than 2 dictionary f
```

```
74 17            JE SHORT de_Atomi.00404734
53               PUSH EBX                        lParam => 0
53               PUSH EBX                        wParam => 0
57               PUSH EDI                        Message => MSG(1004)
FF35 E05D4200    PUSH DWORD PTR DS:[425DE0]      hWnd = NULL
FFD6             CALL ESI                        SendMessageA
83F8 02          CMP EAX,2
7F 07            JG SHORT de_Atomi.00404734
68 E0564100      PUSH de_Atomi.004156E0          ASCII "To use more than 2 dictionary fil
```

ASCII "Sorry, this file can't be cracked with already placed ones simultaneously"

```
74 06            JE SHORT de_Atomi.00406843
B8 545D4100      MOV EAX,de_Atomi.00415D54       ASCII "Fully functional version"
```

وأما هذه Fully Functional version فأنت مخير: إما تعكس القفزة التي فوقها وإما تعكس القفزة التي فوق Free evaluation version

```
74 06            JE SHORT de_Atomi.00406843
B8 545D4100      MOV EAX,de_Atomi.00415D54       ASCII "Fully functional version"
```

أضف أيضاً أنه يمكنك تغيير محتواها بعد عكسها إلى اسمك مثلاً : by AT4RE/heKCEr Fully

أو مباشرة تغير محتواها من Free evaluation version إلى : by AT4RE/heKCEr Fully

```
 v74 06          JE SHORT de_Atomi.00406843
 . B8 545D4100   MOV EAX,de_Atomi.00415D54        ASCII "Fully by AT4RE\heKCEr    "
```

بقيت هنالك مشكلة واحدة، وهي: عندما نضيف أكثر من ملف تظهر هذه الرسالة،

```
ASCII "Sorry, this file can't be cracked with already placed ones simultaneously"
```

ولحل هذه المشكلة، انقر بشكلٍ مزدوج على هذه الرسالة ثم اصعد للأعلى قليلاً إلى أن تصل للعنوان00404FEF وعندها قم بعكس القفزة،

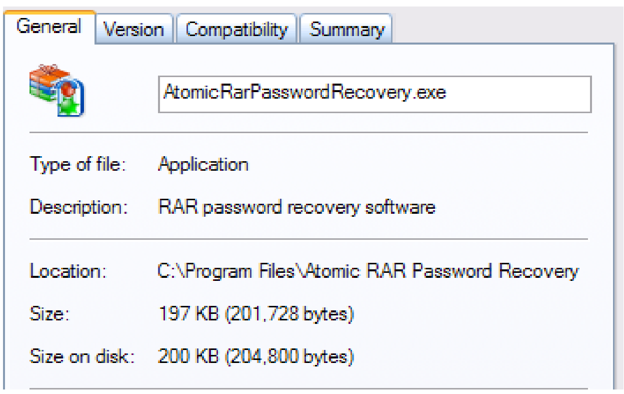
```
 v75 2C          JNZ SHORT de_Atomi.0040501D
```

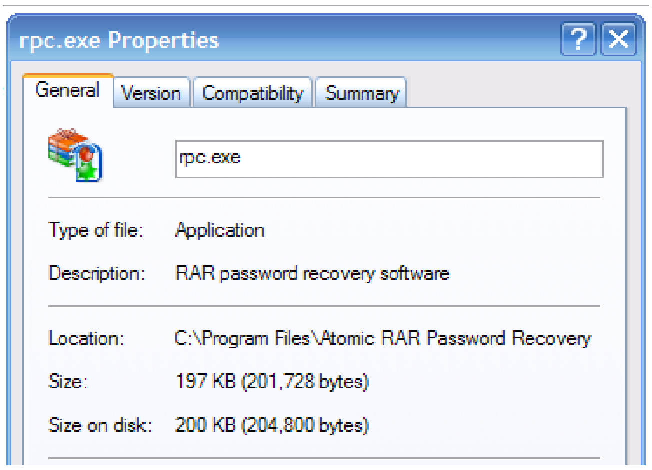
وبهذا أصبح البرنامج كاملاً والكمال لله تعالى.

الآن احفظ الملف المعدل باسم AtomicRarPasswordRecovery.exe وأيضا باسم rpc.exe .. لماذا؟!

لأن rpc.exe نسخة من AtomicRarPasswordRecovery.exe ، وفي بعض الأحيان يتم فتح الــ Project الذي حفظته أنت في rpc.exe ، ولهذا وجب حفظه بهذا الاسم أيضاً.

وبإمكانك التأكد من عدد البايتات :





الآن شغل البرنامج واستمتع بميزات النسخة الكاملة.

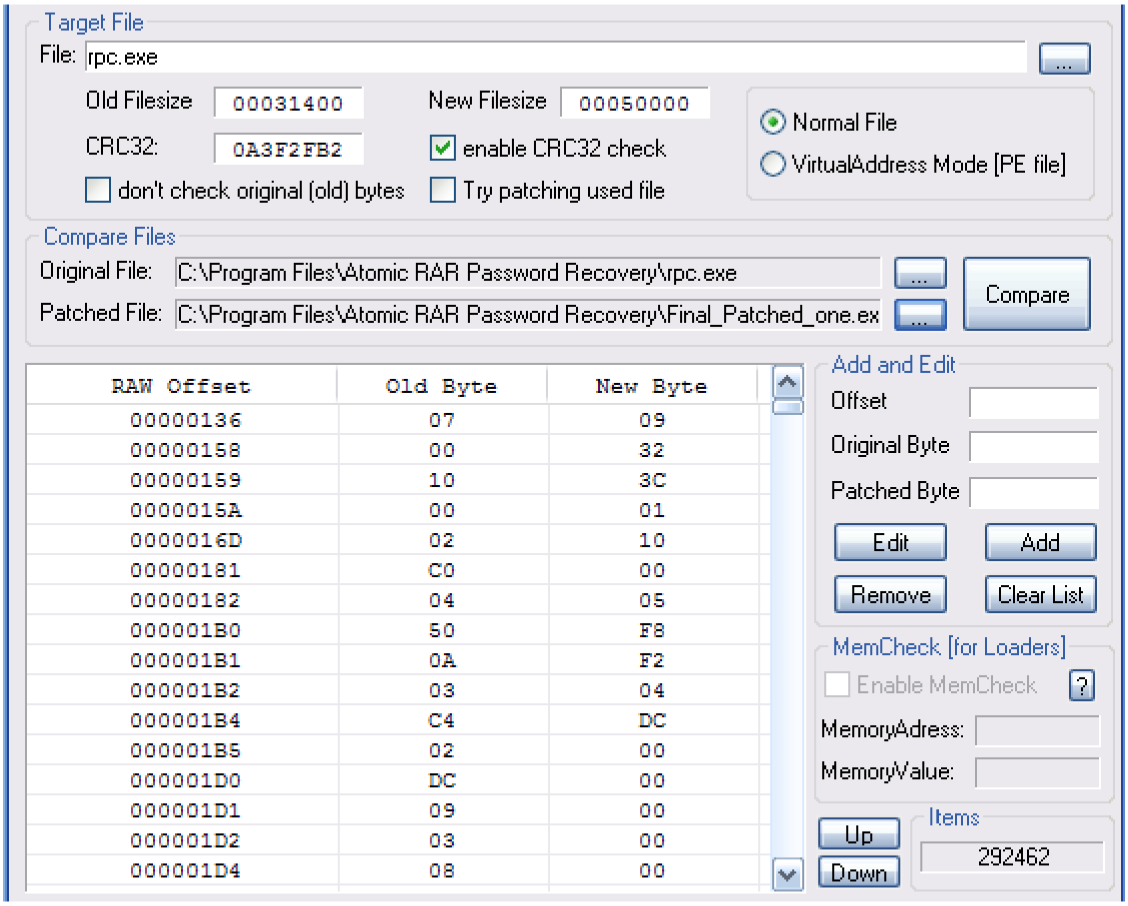
بإمكانك أن تصنع باتش أو لودر أو تعمل Inline Patching وكل ذلك سنتطرق إلى شرحه.

2. عمل CRACK صالح للنشر

2.1 الطريقة الأولى : صنع باتش باستخدام DUP 2.19

افتح واجهة البرنامج الرئيسي ثم اضغط على New Project وقم بتعديلاتك الخاصة، وكالعادة اضغط الزر الأيمن واختر add Offset Patch ومن الأعلى في Target File Tab قم باختيار الملف الأصلي (المشفر)، وفي Compare Files Tab وعند Original File قم باختيار الملف الأصلي المشفر مرة أخرى ، وعند Patched File اختر الملف المفكوك المكسور، ثم اضغط على Compare وعندها سشاهد البايتات قبل استخدام باتشك الخاص وبعد استخدام باتشك الخاص، أي الفرق بين البايتات في الملفين، اضغط save للحفظ.

قم بعمل تلك الخطوات السابقة مرة مع rpc.exe والملف المعدّل، ومرة مع AtomicRarPasswordRecovery.exe والملف المعدّل.



أخيراً قم بالضغط على Create Patch وألف مبارك صنع الباتش.

2.2. الطريقة الثانية : عمل InlinePatch

سنقوم بعمل Inline Patching للبرنامج وذلك بواسطة برنامج ASProtect SKE Inline Patchinger 0.2

ويمكنك تحميله من هنا :



أيضاً سنقوم بكسر البرنامج بطريقة أخرى، وهي الطريقة الأفضل، وذلك بتسجيل البرنامج من البداية دون الحاجة إلى إزالة كل نافذة مزعجة على حدة.

وللوصول إلى هذا المكان السحري الذي سنسجل فيه البرنامج تسجيلاً كاملاً هناك عدة طرق وأسهلها هي الطريقة الآتية :

عند محاولتنا لإزالة النافذة المزعجة مثلاً هذه : "To crack more than 2 files simultaneously you must register the software" سنلاحظ وجود مقارنة هامة جداً :

```
00402FA6  PUSH EBX                                    ┌lParam
00402FA7  PUSH EAX                                    │wParam
00402FA8  PUSH 1013                                   │Message = MSG(1013)
00402FAD  PUSH DWORD PTR DS:[425FF8]                  │hWnd = NULL
00402FB3  MOV DWORD PTR DS:[425F44],EAX               │
00402FB8  CALL NEAR DWORD PTR DS:[<&user32.SendMessageA>] └SendMessageA
00402FBE  CMP BYTE PTR DS:[425AA0],BL
00402FC4  PUSH 3
00402FC6  POP EDI
00402FC7  JNZ SHORT _AtomicR.00403010
00402FC9  CMP WORD PTR DS:[425C70],2
00402FD1  JBE SHORT _AtomicR.00402FDD
00402FD3  PUSH _AtomicR.00415728                      ASCII "To crack more than 2 files simultaneously y
00402FD8  JMP _AtomicR.004032F1
00402FDD  MOV AL,BYTE PTR DS:[426014]
00402FE2  TEST AL,1
00402FE4  JE SHORT _AtomicR.00402FFA
00402FE6  CMP WORD PTR DS:[425DB8],2
00402FEE  JBE SHORT _AtomicR.00402FFA
00402FF0  PUSH _AtomicR.004156E0                      ASCII "To use more than 2 dictionary files you mus
00402FF5  JMP _AtomicR.004032F1
00402FFA  TEST AL,2
00402FFC  JE SHORT _AtomicR.00403010
00402FFE  CMP DWORD PTR DS:[4260A0],EDI
00403004  JBE SHORT _AtomicR.00403010
00403006  PUSH _AtomicR.004153C0                      ASCII "To find passwords which have a dynamic part
0040300B  JMP _AtomicR.004032F1
00403010  MOV EAX,DWORD PTR DS:[425C74]
```

هذه المقارنة الموجودة عند العنوان 00402FBE ولو كانت القيمة الموجودة في 425AA0 تساوي 1 فإن القفزة الموجودة بعدها سوف تنفذ، وبذلك سنتخطى كل النافذة المزعجة، أيضاً هذه المقارنة تتكرر في عدة أماكن من البرنامج؛ حيث يختبر فيها البرنامج

التسجيل فإذا كنت مسجلاً فإن القفزة ستتم ولن يعرض لك أي نافذة مزعجة أو Limitation لذلك سينصب عملنا على وضع القيمة 1 في العنوان 425AA0 دائماً حتى يتم تسجيل البرنامج.

وللوصول إلى المكان المناسب لعمل ذلك ابحث عن الـسترينغ KEYISNOTCOMPATIBLE

```
00406C81  .  833D 54584200 FF  CMP DWORD PTR DS:[425854],-1
00406C88  .v 74 2E             JE SHORT _AtomicR.00406CB8
00406C8A  .  68 B45E4100       PUSH _AtomicR.00415EB4          [s2 = "KEYISNOTCOMPATIBLE"
00406C8F  .  68 A05A4200       PUSH _AtomicR.00425AA0          |s1 = ""
00406C94  .  E8 81CF0000       CALL <JMP.&msvcr71.strcmp>      Lstrcmp
00406C99  .  85C0              TEST EAX,EAX
00406C9B  .  59                POP ECX
00406C9C  .  59                POP ECX
00406C9D  .v 75 19             JNZ SHORT _AtomicR.00406CB8
00406C9F  .  68 385F4100       PUSH _AtomicR.00415F38          ASCII "Your license key is :
00406CA4  .  68 0C5F4100       PUSH _AtomicR.00415F0C          ASCII "RAR Password Cracker
00406CA9  .  6A 10             PUSH 10
00406CAB  .  E8 36FDFFFF       CALL _AtomicR.004069E6
00406CB0  .  83C4 0C           ADD ESP,0C
00406CB3  .  83C8 FF           OR EAX,FFFFFFFF
00406CB6  .v EB 46             JMP SHORT _AtomicR.00406CFE
00406CB8  >  A1 205B4200       MOV EAX,DWORD PTR DS:[425B20]
00406CBD  .  85C0              TEST EAX,EAX
```

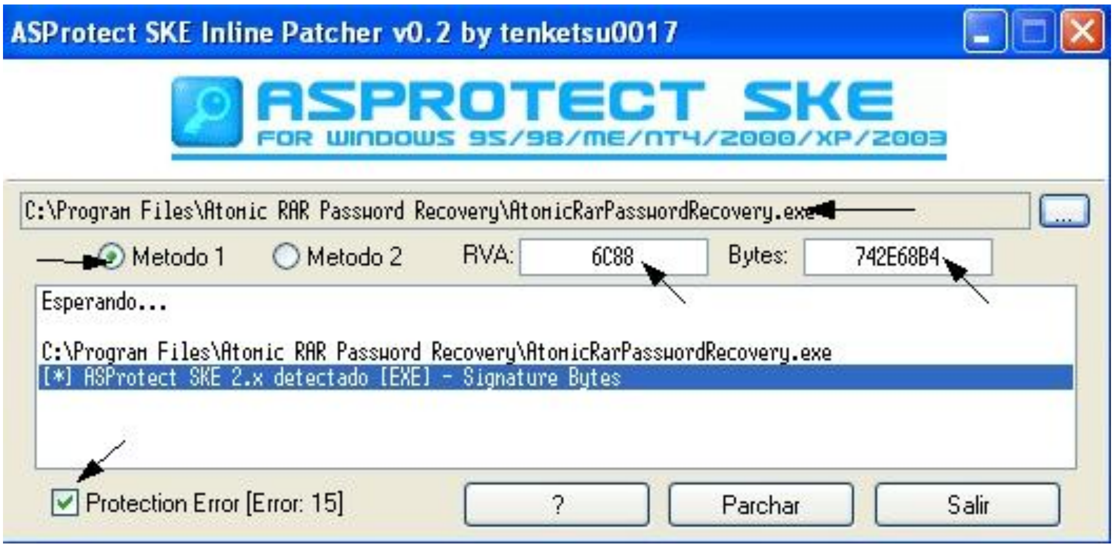
ستلاحظ هنا أن البرنامج يقوم بالتأكد من التسجيل عند بداية تشغيله فإذا كنت غير مسجل يضع 0 في 425AA0.

لعمل InlinePatch سنقوم بإيقاف البرنامج عند العنوان 00406C88 ثم نقوم بوضع 1 في 425AA0 عن طريق الأمر: MOV DWORD PTR DS:[425AA0],1 لذلك احفظ هذه البيانات لأننا سوف نستخدمها لاحقاً :

```
PATCH = MOV DWORD PTR DS:[425AA0],1
RVA = 6C88
BYTES = 742E68B4
```

```
Address   Hex dump          bytes                          ASCII
00406C88  74 2E 68 B4 5E 41 00 68 A0 5A 42 00 E8 81 CF 00  t.h´^A.h ZB.èپ.
```

الآن جاء دور برنامج ASProtect SKE Inline Patchinger :



سندخل البيانات كما في الصورة السابقة ثم نختار الملف، وهو ملف البرنامج المحمي بـــ Asprotect ثم نضغط على Patch :



سوف نجد نسخة جاهزة من البرنامج يمكن عمل InlinePatch لها.

قم بفتح الملف AtomicRarPasswordRecovery.exe بواسطة OllyDbg وستجد أن برنامج الـــ Patcher قد أضاف بعض الأكواد الجديدة لتقوم بتشغيل البرنامج في الذاكرة ثم إيقافه عند العنوان الذي حددناه من قبل 00406C88 لكي نقوم بإدخال الكود : MOV DWORD PTR DS:[425AA0],1 وهو الذي سوف يسجل البرنامج.

انزل للأسفل حتى تجد مجموعة من الـــ NOP وهي مخصصة لإدخال أكواد الباتش :

```
0044C402  .  9D              POPFD
0044C403  .  61              POPAD
0044C404  .  FF35 D0CD4400   PUSH DWORD PTR DS:[44CDD0]
0044C40A  .  C3              RETN
0044C40B  >  FF35 000A4300   PUSH DWORD PTR DS:[<&kernel32.GetModuleH   kernel32.GetModuleHandleA
0044C411  .  FF15 E8CD4400   CALL NEAR DWORD PTR DS:[44CDE8]            AtomicRa.0044C01A
0044C417  .  837C24 28 00    CMP DWORD PTR SS:[ESP+28],0
0044C41C  .^ 75 B7           JNZ SHORT AtomicRa.0044C3D5
0044C41E  .  90              NOP
0044C41F  .  90              NOP
0044C420  .  90              NOP
0044C421  .  90              NOP
0044C422  .  90              NOP
0044C423  .  90              NOP
0044C424  .  90              NOP
0044C425  .  90              NOP
0044C426  .  90              NOP
0044C427  .  90              NOP
0044C428  .  90              NOP
0044C429  .  90              NOP
0044C42A  .  90              NOP
0044C42B  .  90              NOP
0044C42C  .  90              NOP
0044C42D  .  90              NOP
0044C42E  .  90              NOP
0044C42F  .  90              NOP
0044C430  .  9D              POPFD
0044C431  .  61              POPAD
0044C432  .  FF35 000A4300   PUSH DWORD PTR DS:[<&kernel32.GetModuleH   kernel32.GetModuleHandleA
0044C438  .  C3              RETN
```

ثم قم بإدخال الأمر MOV DWORD PTR DS:[425AA0],1 عند أول NOP :

```
0044C41C  .  75 B7              JNZ SHORT AtomicRa.0044C3D5
0044C41E     C705 A05A4200      MOV DWORD PTR DS:[425AA0],1
0044C428  .  90                 NOP
0044C429  .  90                 NOP
0044C42A  .  90                 NOP
0044C42B  .  90                 NOP
0044C42C  .  90                 NOP
0044C42D  .  90                 NOP
0044C42E  .  90                 NOP
0044C42F  .  90                 NOP
0044C430  .  9D                 POPFD
0044C431  .  61                 POPAD
```

بعدها قم بحفظ الملف الجديد وهو الملف المكسور بـــ InlinePatch.

بكل سهولة تستطيع الآن عمل Patch للبرنامج بواسطة برنامج Dup.

إن ميزة InlinePatch رائعة جداً؛ حيث ستحصل من خلالها على باتش صغير الحجم مقارنةً فيما لو قمت بفك الحماية عن الملف وكسره.

2.3 الطريقة الثالثة : صنع لودر باستخدام برنامج DUP 2.17

كما فعلنا في الـــ Inline Patching سنقوم بعمل لودر يكتب في الذاكرة هذا الأمر :

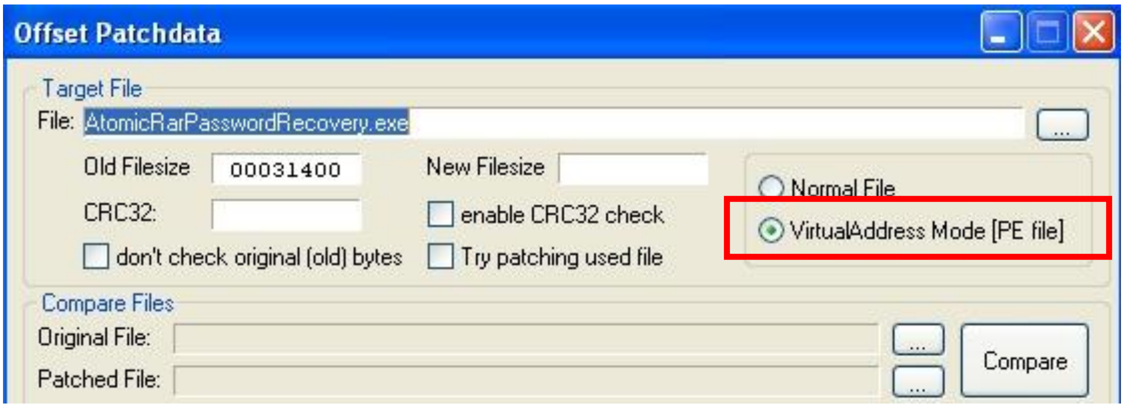
MOV DWORD PTR DS:[425AA0],1

وهذا الأمر هو الذي سيسجل البرنامج، ويمكنك معرفة البايتات من خلال الصورة السابقة بعد كتابتنا للأمر :

```
00406CB8 = A1 => C7
00406CB9 = 20 => 05
00406CBA = 5B => A0
00406CBB = 42 => 5A
00406CBC = 00 => 42
00406CBD = 85 => 00
00406CBE = C0 => 01
00406CBF = 74 => 00
00406CC0 = 15 => 00
00406CC1 = 68 => 00
00406CC2 = 00 => 90
00406CC3 = 5A => 90
00406CC4 = 42 => 90
00406CC5 = 00 => 90
```

الجدول السابق يبين العنوان + البايت القديم + البايت الجديد (Patched)

افتح برنامج DUP :



أدخل البيانات الموجودة في الجدول عن طريق Add and Edit ولا تنس اختيار VirtualAddress mode ، ثم اضغط على Save ثم Create loader ، وبذلك سيتم إنشاء اللودر الذي ستقوم بوضعه جانب ملف البرنامج ومن ثم تشغله ليقوم بتحميل البرنامج وإجراء التغييرات اللازمة ليصبح البرنامج مسجلاً بشكلٍ كامل.

لا تنس عمل لودر لملف rpc.exe .

حمل من المرفقات ملف مشروع dup patcher الخاص بالبرنامج لمن أراد تحميله مباشرة لـــ dup.	

الباب الثالث: الهاشينغ وخوارزميات التشفير

مقدمة في الهاشينغ وخوارزميات التشفير ومكتباتها

Introduction to Cryptology

سنبدأ بمقدمة عن الهاشينغ وخوارزميات التشفير, ثم نسقط المعلومات على مثال عملي يستخدم خوارزمية.

و سيشتمل هذا الشرح على النقاط التالية :

	الملفات اللازمة لمتابعة الدرس هنا.

I. أولاً : لمحة عن مكتبات التشفير وكيفية استخدامها في لغات البرمجة

تستخدم خوارزميات التشفير عموماً لتشفير البيانات وذلك بهدف إضافة طبقة من الحماية على هذه البيانات التي يتم التعامل معها داخل البرنامج أو عبر النقل الإلكتروني في الويب.

وهنا يجب توضيح الفرق بين مفهومين أساسيين :

I.3 المفهوم الأول : خوارزمية التشفير

و هي عبارة عن سلسلة من العمليات تطبق على البيانات لتمويهها. وأهم ما تتميز به :

أنها ذات اتجاهين : اتجاه تشفير البيانات (Encryption) واتجاه فك تشفير البيانات (Decryption)

أي أن الخوارزمية : تقوم بالتشفير ثم يمكنها فك تشفير البيانات وإرجاعها إلى أصلها.

ومن أمثلتها نذكر : RSA , DES , Mars , Blowfish

1.4 المفهوم الثاني : الهاش

وهو يستخدم للتشفير فقط، وليس له اتجاه نحو فك التشفير، ويستخدم لفك تشفير ما يعرف بـــ : Brute Force

حيث يستخدم جداول خاصة ويقوم بتشفير عدد هائل من العبارات ويطابق ناتج التشفير مع الكلمة المشفرة التي تحث عن فك تشفيرها.

و من أمثلته : MD4 , MD5 , SHA , Haval

1.5 المكتبات المستخدمة في خوارزميات التشفير والهاشينغ

قام عدد من المبرمجين المحترفين بإنشاء مكتبات لكلٍ من خوارزميات التشفير والهاشينغ، وذلك بهدف تسهيل المهمة على المبرمجين في استخدام كلا النوعين السابقين.

و قد توفرت هذه المكتبات لأغلب لغات البرمجة, ويكفيك فقط استدعاء الإجراء أو التابع والمكتبة تتكفل بالباقي وتعطيك الناتج .

مثلاً : لو أردت أن أشفر كلمة Hello بواسطة خوارزمية DES ، فأقول مجازاً :

شفر لي باستخدام خوارزمية DES الكلمة Hello , واستخدم في التشفير المفتاح 123.

في الحقيقة هذا هو التعبير المجازي، ولإسقاطه على الكتابة البرمجية نقول :

EnCryptString (Hello, 123)

– فكلمتي Hello و 123 تدعى بالباروميترات وهي التي يحتاج إليها الإجراء EnCryptString لينجز عمله.

1.6 المكتبات ولغات البرمجة

1.6.1 كيف تأتي هذه المكتبات؟

في الحقيقة تختلف لاحقة المكتبة ومحتوياتها حسب اللغة البرمجية المستعملة، وعلى سبيل المثال وليس الحصر وليكن مكتبة RSA :

– Visual Basic 6 : RSA.bas أو RSA.Cls

– Assembly : RSA.asm

RSA.pas : Delphi –

RSA.inc : C, C++ –

هذا شكل من أشكال الملفات لكل لغة، وقد يتم تضمين المكتبات في وحدات DLL ويستدعيها المبرمج بواسطة الـــ Export Tables.

إن استخدام المكتبات وتطويعها يختلف حسب مهارتك البرمجية وحدود اللغة التي تستخدمها.

إذا فتحت هذه المكتبة باللغة التي تستخدمها ستعثر على الإجراء أو التابع الذي يحقق لك ما تهدف إليه.

لنأخذ على سبيل المثال المكتبة DES ولنفتحها في دلفي، سنجد الإجراء التالي :

```
Function EncryStr (Str, Key: String): String;
```

EncryStr : اسم التابع، واسمه يشير إلى عملية التشفير لسلسلة نصية.

Str : الكلمة المراد تشفيرها.

Key String : مفتاح التشفير، ويطلب منك التابع أن يكون من النوع النصي String ولا يمكن أن يكون رقمياً Integer .

String : أي أن القيمة الناتجة عن هذا التابع هي من النوع النصي.

فإذا أردت تشفير الكلمة Hello بمفتاح التشفير RobenEDH56 ووضع الناتج في المربع النصي Text1 فاكتب :

```
Text1.Text := EncryStr ( Hello , RobenEDH56 ) ;
```

يأتي السؤال هنا : كيف أعرف ما هو التابع الذي سأستخدمه؟ وما هي باراميتراته؟

على سبيل المثال هذا الأجراء :

```
Procedure RSAEncrypt (P : String; Var exp, modb : TFGInt; Var E : String) ;
```

من الواضح أنه لتشفير عبارة باستخدام RSA ، ولكن كيف لي أن أعرف ما هي البراميترات التي سأمررها له؟

على سبيل المثال : modb : TFCInt ما هو؟ وما هو النوع المعرف به وكيف أستخدمه؟

أمامك ٥ نا الخيارات التالية :

1) قراءة التعليمات (Help) المرفقة مع المكتبة.

2) الاطّلاع على مثال يستخدم هذه المكتبة.

3) قراءة ثوابت ومتغيرات خوارزمية التشفير.

4) البحث في الإنترنت عن معلومات تفيد في استعمال هذه المكتبة بعد أن تكون قد قمت بالبند (3).

2.6.1. استعمال المكتبة في المشروع البرمجي :

يجب عليك باختلاف أنواع لغات البرمجة أن تربط المكتبة بمشروعك؛ لكي تستطيع استدعاء التابع أو الإجراء منها.

طبعاً لن يشمل حديثنا مكتبات DLL ، ولن نستطيع كذلك التكلم عن جميع الطرق.

أمثلة :

Delphi : يمكن أن تصرح عن اسمها في القسم Uses

مثال تضمين المكتبة Des.pas :

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, Des, StdCtrls;
```

Visual Basic 6: يمكن أن تقوم بإضافة الموديل Des.bas أو الصنف Des.Cls إلى مشروعك.

Assembly : يمكن أن تصرح عنها كما في الشكل التالي :

```
.386
.model flat, stdcall  ;32 bit memory model
option casemap :none  ;case sensitive
include Des.asm
…
…
.code
       …
```

2. ثانياً : التحليل العام للتمرين

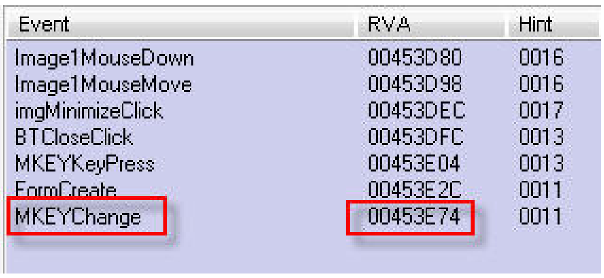
لدى فحص التمرين ببرنامج PEID وأحد إضافاته المساعدة، وهي Kanal نجد :

- البرنامج مبرمج بلغة Delphi وهو غير مضغوط أو مشفر.
- البرنامج يستخدم خوارزميات التشفير التالية :
  - ← DES .
  - ← Rijndael .

. BigNumber ←



ولدى استكشاف البرنامج ببرنامج DEDE الخاص بالبرامج المبرمجة بالدلفي، وكون التمرين لا يحوي على أزرار، فهو يقارن بين السريال المدخل والسريال الصحيح عند الحدث MKEYChange ، ودخلنا إليه وتأكدنا من ذلك وأخذنا عنوان البداية لهذا الإجراء :



3. ثالثاً : تحليل الكود من خلال برنامج OllyDbg

– نحمل التمرين داخل برنامج OllyDbg وندخل المعلومات كما في الصورة التالية :



– ثم نقوم بوضع نقطة توقف عند بداية الإجراء MKEYChange عند العنوان 00453E7 ونحاول تغيير السريال :

يتوقف المنقح هنا ... لكن لنتابع معاً عملية التتبع :

00453E74 PUSH EBP	بداية الإجراء MKEYchange
.... 00453E95 LEA EDX, [LOCAL.9] 00453E98 MOV EAX, DWORD PTR DS:[EBX+2FC] 00453E9E CALL 0043138C	
00453EA3 CMP [LOCAL.9], 0	مقارنة طول الاسم مع الصفر
00453EA7 JE 00453FE2 00453EAD LEA EDX, [LOCAL.10] 00453EB0 MOV EAX, DWORD PTR DS:[EBX+30C] 00453EB6 CALL 0043138C	
00453EBB CMP [LOCAL.10], 0	مقارنة طول السريال المدخل مع الصفر
00453EBF JE 00453FE2 00453EC5 LEA EAX, [LOCAL.7] 00453EC8 CALL 00403E88 00453ECD LEA EAX, [LOCAL.1	
00453ED0 MOV EDX, 454034 ; ASCII "RobenCrackMe1ForElite01"	هذا النص الذي نشاهده هنا هو مفتاح التشفير للخوارزمية DES وهنا فقط يتم تعريفه كثابت ليستخدم كباروميتر عند استدعاء الخوارزمية
00453ED5 CALL 00403F20 00453EDA LEA EDX, [LOCAL.11] 00453EDD MOV EAX, DWORD PTR DS:[EBX+2FC] 00453EE3 CALL 0043138C	
00453EE8 MOV EAX, [LOCAL.11]	هنا يتم دفع الاسم للمكدس
00453EEB LEA EDX, [LOCAL.6]	
00453EEE CALL 00453CEC	هذا الاستدعاء : يقوم بتحويل الاسم إلى النظام Hex ماعدا الحرف الأخير، ففي حالتنا : [Roben] = R[52] + o[6F] + b[62] + e[65] = 526F6265

إذا دخلنا إلى الاستدعاء نلاحظ حلقة التحويل :

```
00453D20  JLE    SHORT 00453D4B
00453D22  MOV    ESI, 1
00453D27  LEA    ECX, [LOCAL.3]              <<< Loop : Convert Name Letters To Hex
00453D2A  MOV    EAX, [LOCAL.1]              >>>> Without Last Letter
00453D2D  MOVZX  EAX, BYTE PTR DS:[EAX+ESI-1
00453D32  MOV    EDX, 2
00453D37  CALL   00407F14
00453D3C  MOV    EDX, [LOCAL.3]
00453D3F  LEA    EAX, [LOCAL.2]
00453D42  CALL   00404150
00453D47  INC    ESI
00453D48  DEC    EBX
00453D49  JNZ    SHORT 00453D27
```

```
00453EF3    LEA     ECX, [LOCAL.5]
```

```
00453EF6    MOV     EDX, [LOCAL.1]
00453EF9    MOV     EAX, [LOCAL.6]
00453EFC    CALL    004527AC
```

هنا يتم دفع المفتاح

هنا دفع ناتج تحويل الاسم إلى hex ماعدا الحرف الأخير

هنا يتم تطبيق خوارزمية التشفير DES

يجب الانتباه هنا إلى أن ناتج خوارزمية تشفير DES يكون في القاعدة 64 (Base 64) وهو في حالتنا :

{$-ËŸ°?÷ (طبعاً مقلوبة من اليمين إلى اليسار)

```
00453F01    LEA     EAX, [LOCAL.5]
00453F04    MOV     EDX, 454054
00453F09    CALL    00404150
00453F0E    LEA     EAX, [LOCAL.2]
```

ما نشاهده هنا هو مفتاح التشفير الخاص بالخوارزمية Rijndael

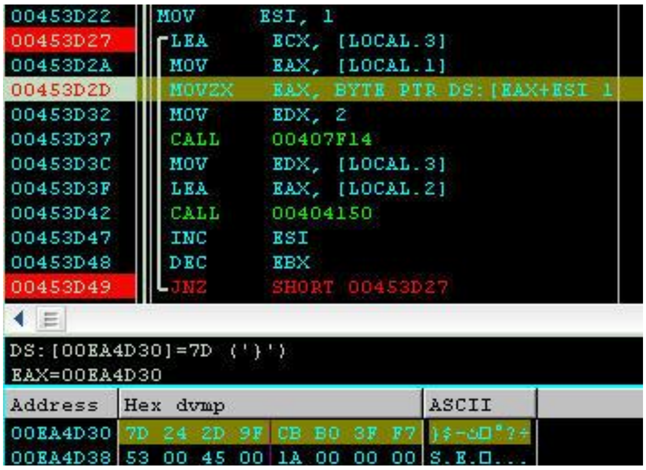
و هنا أيضا : يتم دفعه كثابت ليستخدم كباروميتر عند استدعاء خوارزمية التشفير.

```
00453F11    MOV     EDX, 454060          ; ASCII "6184AE00CD09FA7E73F41D310D1FF48D"
```

```
00453F16    CALL    00403F20
00453F1B    LEA     EDX, [LOCAL.8]
00453F1E    MOV     EAX, DWORD PTR
DS:[EBX+30C]
00453F24    CALL    0043138C
00453F29    LEA     EDX, [LOCAL.14]
00453F2C    MOV     EAX, [LOCAL.5]
```

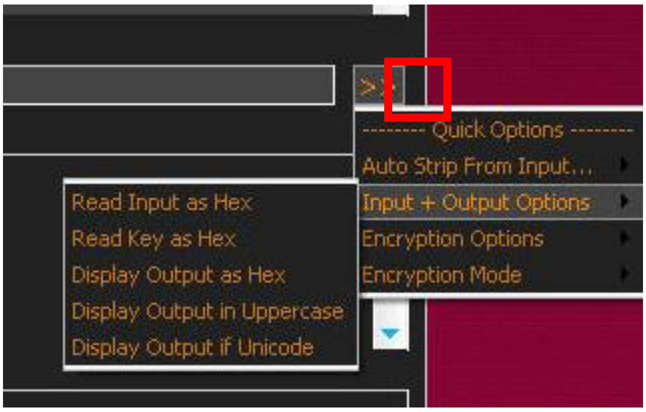
– هنا يتم دفع ناتج خوارزمية DES : لتحويله من القاعدة Base 64 إلى النظام الستة عشري

00453F2F CALL 00453CEC



إذا دخلنا إلى هذا الاستدعاء سنلاحظ حلقة التحويل التالية :

و الناتج هو : **7D242D9FCBB03FF7** كما تلاحظون.



للتأكد :

- أدخل القيمة : **526F6265** في برنامج : SND Reverser Tool في الحقل Input
- اختر خوارزمية التشفير DES ومفتاح التشفير : **RobenCrackMe1ForElite01**
- اختر UPPER CASE من الـ Option

```
00453F34    MOV     EAX, [LOCAL.14]   :
00453F37    LEA     EDX, [LOCAL.13]
00453F3A    CALL    0040797C
00453F3F    MOV     EAX, [LOCAL.13]
00453F42    LEA     ECX, [LOCAL.12]
00453F45    MOV     EDX, [LOCAL.2]
```

00453F48   CALL   00453828

تشفير القيمة : **7D242D9FCBB03FF7** بخوارزمية التشفير Rijndael

```
00453F4D    MOV     EAX, [LOCAL.12]
00453F50    LEA     EDX, [LOCAL.3]
00453F53    CALL    004529C8
00453F58    LEA     ECX, [LOCAL.15]
00453F5B    MOV     EDX, [LOCAL.2]
00453F5E    MOV     EAX, [LOCAL.8]
```

00453F61   CALL   00453828

تشفير القيمة : السريال المدخل **(1234567890)** بخوارزمية التشفير Rijndael

```
00453F66    MOV     EAX, [LOCAL.15]
00453F69    LEA     EDX, [LOCAL.4]
00453F6C    CALL    004529C8
00453F71    MOV     EAX, [LOCAL.3]
00453F74    MOV     EDX, [LOCAL.4]
```

00453F77   CALL   00404294

المقارنة بين ناتج التشفيرين بخوارزمية Rijndeal

هنا يستوقفنا أمر وهو أنه : إذا كان السريال المدخل = ناتج Des + Hex للاسم

- فهذا يعني أن ناتج تشفيرهم بخوارزمية Rijndeal سيكون متساوٍ (لأن القيمتين متساويتين ويتم تشفيرهما بنفس المفتاح) .
- وبالتالي فإن خوارزمية Rijndael وضعت فقط حتى يتم تمويه المقارنة بين السريال الصحيح والسريال المدخل.

00453F7C   JNZ   SHORT 00453FBF

القفز محقق إلى العبارة الخاطئة عند عدم التساوي

```
00453F7E    LEA     EDX, [LOCAL.17]
00453F81    MOV     EAX, DWORD PTR DS:[EBX+2FC]
00453F87    CALL    0043138C
00453F8C    MOV     ECX, [LOCAL.17]
00453F8F    LEA     EAX, [LOCAL.16]
00453F92    MOV     EDX, 45408C                     ;  ASCII "Registered To : "
00453F97    CALL    00404194
00453F9C    MOV     EDX, [LOCAL.16]
00453F9F    MOV     EAX, DWORD PTR DS:[EBX+308]
00453FA5    CALL    004313BC
00453FAA    MOV     EAX, DWORD PTR DS:[EBX+308]
00453FB0    MOV     EAX, DWORD PTR DS:[EAX+68]
00453FB3    MOV     EDX, 8000
00453FB8    CALL    0041C648
00453FBD    JMP     SHORT 00453FE2
00453FBF    MOV     EDX, 4540A8                            ;  ASCII "UnRegistered"
```

4. رابعاً : تحليل خوارزمية حساب السريال للاسم

إذن مما سبق نستنتج خوارزمية حساب السريال وفق الخطوات التالية :

1) طول الاسم يجب أن يكون حرفين أو أكثر (لأن البرنامج يحذف الحرف الأخير عند التحويل إلى Hex).

2) تشفير الناتج بواسطة خوارزمية DES مع المفتاح : **RobenCrackMe1ForElite01** .

3) تحويل ناتج التشفير إلى Hex.

4) تحويل الناتج إلى أحرف كبيرة Upper Case.

هذه الخوارزمية لحساب السريال الصحيح، ويمكن أن تسقطها على أي لغة برمجة تجيدها.

5. خامساً : كتابة الكيجن

سنستخدم هنا لغة Delphi 7 وسنكتب الشيفرة الخاصة بتحصيل السريال (السورس كود مرفق) :

```
Procedure TForm1.GetSerialClick(Sender: TObject);
 Var
   I :Integer ;
   UName,HexName, Key, StrDES, StrHEX: String;
 begin
  StrHEX:='';
  UName := TXTName.text ;
  If Length(UName ) < 2 Then                    { التحقق من طول الاسم المدخل }
  Begin
  TXTSerial.Text := ' Name Must Be >= 2 ';
```

```
   Exit;
   End;
   For I:=1 To ( Length(UName)-1 ) do
   Begin
     HexName:= HexName + IntToHex(Ord(UName[I]),2); {تحويل الاسم إلى نظام الستة عشري ماعدا الحرف الأخير
}
   End;
   Key:= 'RobenCrackMe1ForElite01';
   StrDES:= EncryStr( HexName, Key );             { تشفير الناتج }
   For I:=1 To Length(StrDES) do
   Begin
   StrHex:= StrHex + IntToHex(Ord(StrDES[I]),2); { تحويل الناتج إلى هيكس }
   End;
   TXTSerial.Text := UpperCase (StrHex);         { تحويل إلى أحرف كبيرة }
End;
```

خوارزمية الجمل

ElGamal Algorithm

1. مقدمة

نقدم هنا شرحاً مبسطاً لخوارزمية التشفير الجمل ElGamal من خلال مثال عملي، ثم سنتطرق لكيفية كتابة كيجن للتمرين.

وسيشتمل هذا الشرح على النقاط التالية :

2. أولاً : لمحة موجزة عن خوارزمية التشفير ElGamal

2.1 معلومات عامة

- تم تطوير خوارزمية الجمل (ElGamal) في مختبرات Hewlett –Packard Labs في كاليفورنيا في الولايات المتحدة الأمريكية على يد العالم المصري "طاهر الجمل" Taher ElGamal ولذلك سميت الخوارزمية باسمه.
- تندرج خوارزمية الجمل تحت عنوان كبير: يعرف باسم: مخطط التشفير بالمفتاح العام Public-Key Encryption Scheme . وتستخدم كقاعدة لها: Discrete algorithm problem & Diffie-Hellman Problem .
- نعتقد أن المصطلحين الأخيرين بحاجة إلى اختصاصي في العلوم الرياضية لشرحهما، وهذا ليس جوهر موضوعنا.
- الأمر المهم في هذه الخوارزمية هي استخدامها لمخطط التوقيع الرقمي، والذي يعرف بالاسم: Digital Signatures Scheme . وهذه تستخدم لحمل الطابع الشخصي للشخص الذي قام بالتشفير، وللمنع من العبث بالرسالة المشفرة. هذا المصطلح بشرحه السابق ينطبق بشكل أكبر على عمليات النقل الإلكتروني عبر الويب.

2.2. شرح ثوابت الخوارزمية العامة والخاصة

كما سبق وذكرنا فإن الخوارزمية تندرج تحت بند: خوارزميات التشفير بالمفتاح المعلن أو العام.

و هذا يعني أن الخوارزمية تستخدم توليفة من المفاتيح العامة (Public) والخاصة (Private).

لنبدأ الآن بتعريف ثوابت الخوارزمية دون الدخول في المعادلات الرياضية :

الرمز (P) : وهو مفتاح عام (أي أنه معلن وموجود داخل البرنامج) ويجب أن يكون هذا المفتاح عبارة عن عدد أولي وكبير نوعاً ما؛ ليحقق تشفير متين للبيانات.

الرمز (Y) : مفتاح عام (أي أنه معلن وموجود داخل البرنامج) ويدعى : *Power* .

الرمز (G) : مفتاح عام (أي أنه معلن وموجود داخل البرنامج) ويدعى : *Base* .

الرمز (X) : المفتاح الخاص (وله أهمية بالغة في فك التشفير ومهم جداً في حساب التواقيع الرقمية).

الرمزين (a , b) : ويدعيان بالتواقيع الرقمية وهما المسؤولان عن التأكد من الرسالة المشفرة، ويتم حسابهم اعتماداً على المفتاح الخاص (X) .

الرمز (K) : رقم عشوائي غالباً يحمل القيمة 1 ويدعى : *Passphrase* .

- إذا من خلال العرض السابق للثوابت : نجد أن القيمة الوحيدة المجهولة هي : المفتاح الخاص X و a, b يتم حسابها من خلاله.
- لإيجاد هذا المفتاح اعتماداً على القيم المتوفرة لدينا نحن بحاجة إلى تطبيق المعادلة التالية والمجهول الوحيد فيها هو X :

$$Y = G \wedge X \ Mod \ P$$

- هناك عدد من البرمجيات التي تحسب لك قيمة X بناءً على المعادلة السابقة، وطبعاً الأمر يستغرق وقتاً أكبر كلما كبرت قيمة هذه القيم (P,G,Y).

و من هذه البرمجيات نذكر : **DLP Tool** (Discrete Logarithm Problem)

وهناك تطبيق على الإنترنت من خلال الموقع التالي :



3. ثانياً : لمحة موجزة عن مكتبة FGInt

3.1 معلومات عامة

- تعتبر المكتبة FGInt ووحداتها البرمجية المرفقة معها ملاذاً جيداً لمبرمجي دلفي لكتابه تطبيقات تستخدم بعض خوارزميات التشفير والهاشينغ.
- لا يهدف شرحنا إلى تناول المكتبة بجميع إمكانياتها، بل سنركز على الأمور التي تخصنا في درسنا هذا.
- تأتي المكتبة مع الملفات التالية :

```
FGInt.pas - FGIntElGamal.pas - FGIntPrimeGeneration.pas
```

بالإضافة إلى العديد من الملفات التي لا تهمنا هنا.

- في درسنا هذا سنستخدم الملفين الأول والثاني، أما الثالث (FGIntPrimeGeneration.pas) فهو يستخدم لتكوين المفاتيح العامة والخاصة، مع أن هنالك بعض البرمجيات التي تقوم بهذا الأمر وأنت تستخدم القيم جاهزة (مرفق مع الدرس البرنامج : ElGamal والذي يوجد لك القيم السابقة)

3.2 التعرف على إجرائيات الوحدة FGIntElGamal.pas

- لنقم بفتح الوحدة من خلال بيئة التطوير دلفي (من لا يملك دلفي يمكنه فتحها بأي محرر نصي) .
- لنتابع الآن الإجرائيات المهمة في درسنا هذا :

```
Procedure ElGamalEncrypt  (P : String; Var g, y, k, modp : TFGInt; Var E : String);
Procedure ElGamalDecrypt  (E : String; Var x, p : TFGInt; Var D : String);
Procedure ElGamalSign     (M : String; Var p, g, x, k : TFGInt; Var a, b : String);
Procedure ElGamalVerify   (Var g, y, p : TFGInt; a, b, M : String; Var ok : Boolean);
```

- إذن هذه هي الإجرائيات المستخدمة في المكتبة، و يهمنا منها في هذا الدرس الثالث والرابع :
  - يستخدم الإجراء الثالث ElGamalSign في إيجاد التواقيع الرقمية a,b والمجهول الوحيد هنا هو X .
  - يستخدم الإجراء الرابع ElGamalVerify في التحقق من صحة الرسالة المشفرة اعتماداً على التواقيع الرقمية a,b .

سنغوص قليلاً في الإجراء الرابع **ElGamalVerify** :

من تعريف الإجراء نجد أنه يستخدم البارومترات التالية :

G , Y , P : الذين عرفناهم في المقدمة، ولكن انتبه إلى أنه يعرفهم على أنهم من الصنف TFGInt ولكن ماهو الصنف TFGInt ؟

في الحقيقة هو صنف تفهمه وتتعامل معه المكتبة FGInt فأنت تقدم لها الرقم عشري Decimal وهناك توابع تحوله إلى الصنف TFGInt ، ولفهم أكثر لهذه العملية دعونا نلقي نظرة على الوحدة FGInt.pas :

```
…
…
Procedure ConvertBase256StringToHexString (Str256 : String; Var HexStr : String);    {1}
Procedure ConvertHexStringToBase256String (HexStr : String; Var Str256 : String);    {2}
…
Procedure Base10StringToFGInt (Base10 : String; Var FGInt : TFGInt);                 {3}
```

– الإجراءات {1} و {2}: تستخدم للتحويل بين القاعدة 256 والقاعدة 16.

– الإجراء {3}: يستخدم للتحويل من القاعدة العشرية إلى الصنف TFGint وهي ما سنستخدمه في كتابة كيجن لهذا التمرين.

- لنتابع مع الإجراء ElGamal Verify :
  - ← انتبه أن خرج هذا الأجراء (OK) هو قيمة منطقية (نعم ولا) ، (True , False).
  - ← إذا كانت التواقيع الرقمية صحيحة من اجل الرسالة المشفرة M فالناتج هو نعم أو True أو 1
  - ← إذا كانت التواقيع الرقمية غير صحيحة من أجل الرسالة المشفرة M فالناتج هو لا أو False أو 0.
- لننزل للأسفل قليلاً في الوحدة FGIntElGamal.pas :
  - ← سنلاحظ تعليمات الإجراء **ElGamalSign** :
  - ←

```
Procedure ElGamalSign(M : String; Var p, g, x, k : TFGInt; Var a, b : String);
…
   FGIntToBase256String(temp1, a);
…
   FGIntToBase256String(temp3, b);
End;
```

ونلاحظ هنا أن البارومترات المطلوبة هي (P,G,X,K) وكلها معلومة لدينا ماعدا X مطلوب حسابه.

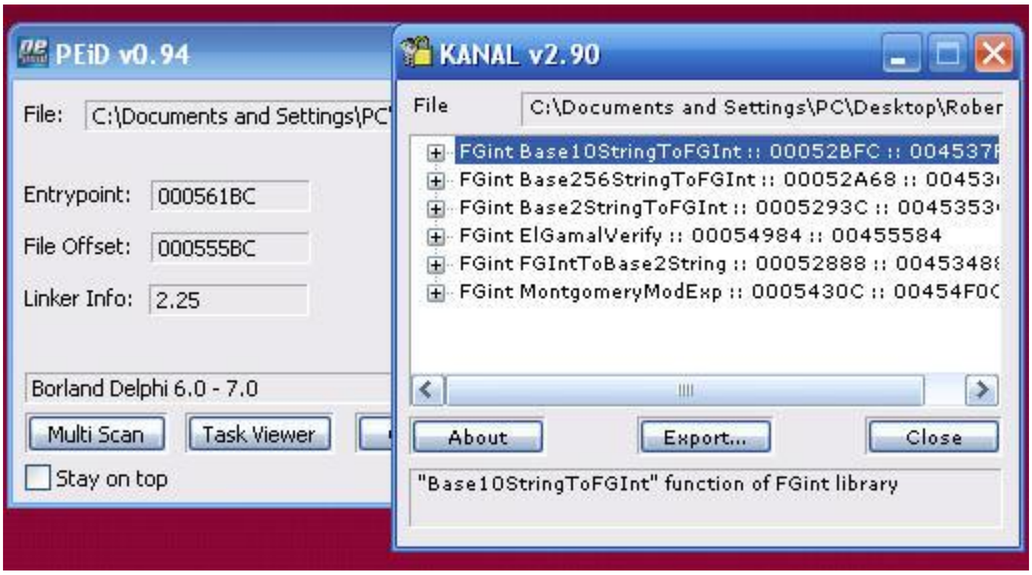
كما نلاحظ أن خرج هذا التابع يكون في القاعدة 256 (Base 256).

4. ثالثاً : التحليل العام للتمرين

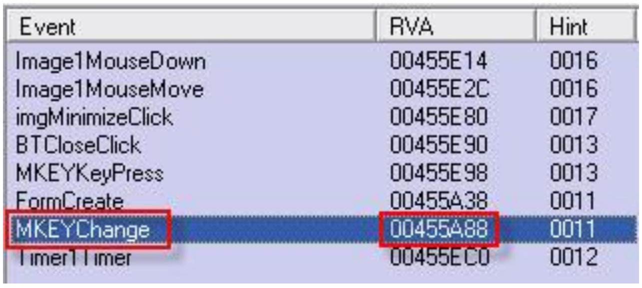
- لدى فحص التمرين ببرنامج PEID وأحد إضافته المساعدة وهي Kanal سنجد :
- البرنامج مبرمج بلغة Delphi وهو غير مضغوط أو مشفر.
- البرنامج يستخدم توابع تشفير عديدة، ومن خلال المقدمة السابقة نجد أن أهمها ما يلي :

Base 10 To FGInt ←

FGIntElGamalVerify ←



ولدى استكشاف البرنامج ببرنامج DEDE ، وكون التمرين لا يحتوي على أزرار فهو يقارن بين السريال المدخل والسريال الصحيح عند الحدث **MKEYChange** ، ودخلنا إليه وتأكدنا من ذلك وأخذنا عنوان البداية لهذا الأجراء :



Event	RVA	Hint
Image1MouseDown	00455E14	0016
Image1MouseMove	00455E2C	0016
imgMinimizeClick	00455E80	0017
BTCloseClick	00455E90	0013
MKEYKeyPress	00455E98	0013
FormCreate	00455A38	0011
MKEYChange	00455A88	0011
Timer1Timer	00455EC0	0012

5. رابعا : تحليل الكود من خلال برنامج OllyDbg

– نحمل التمرين داخل برنامج OllyDbg وندخل المعلومات كما في الصورة التالية :



– ثم نقوم بوضع نقطة توقف عند بداية الأجراء MKEYChange عند العنوان **00455A88** ونحاول تغيير السريال:

يتوقف المنقح هنا، لكن دعنا نتابع معاً عملية التتبع ونقتصر فقط على النقاط الهامة :

00455A88 PUSH EBP …. 00455AE9 DEC EAX 00455AEA JGE SHORT 00455B45 ….	;بداية الإجراء ; ;مقارنة طول الاسم مع الصفر
00455B5B CMP EAX, **10** 00455B5E JE SHORT 00455BB9 …	;مقارنة طول السريال مع 16
00455BB9 LEA EAX, [LOCAL.7]	; " Parameters : G "
00455BBC MOV EDX, 455DC4	; ASCII "878977914"
…….	
00455BCC CALL 004537FC	; Convert Base 10 to TFGInt
00455BD1 LEA EAX, [LOCAL.8]	; " Parameters : Y "
00455BD4 MOV EDX, 455DD8	; ASCII "972424840"
……	
00455BE4 CALL 004537FC	; Convert Base 10 to TFGInt
00455BE9 LEA EAX, [LOCAL.9]	; "Parameters : P "
00455BEC MOV EDX, 455DEC	; ASCII "4150976723"
…. .	
00455BFC CALL 004537FC	; Convert Base 10 to TFGInt
…. 00455C16 MOV ECX, 8 00455C1B MOV EDX, 1 00455C20 CALL 004043A8	أخذ أول 8 أحرف من السريال
…. .	
00455C3D CALL 0043242C	; Convert Hex To Base 256
…. 00455C45 MOV ECX, **10** 00455C4A MOV EDX, **9**	
00455C4F CALL 004043A8	;أخذ آخر 8 أحرف من السريال
…. . 00455C68 CALL 0043242C ….	; Convert Hex To Base 256

00455C7C PUSH EAX … ….	; Push Name: ' Dr.RobenX '
00455C86 CALL 00455584	; ElGamalVerify
00455C8B CMP BYTE PTR SS:[EBP 31], 1	; IF ok = True (1) Then Registered
00455C8F JNZ SHORT 00455CEF	
……	
00455CA5 MOV EDX, 455E00	; ASCII " Registered To : "
……	
00455CEF MOV EDX, 455DA4	; ASCII "Unregistered Version"

6. خامسا: تحليل الثوابت وإيجاد المفتاح الخاص بالخوارزمية

- حتى نكون واقعيين في التحليل فيجب التنبيه إلى أن التتبع في المنقح لا يكفي ما لم يمكن عندك خلفية جيدة في ثوابت الخوارزمية وإجرائيات المكتبة FGIntElGamal وقواعد القيم التي تتعامل معها.
- فلو أخذنا الأجراء ElGamalVerify :

```
Procedure ElGamalVerify (Var g, y, p : TFGInt; a, b, M : String; Var ok : Boolean);
```

و أسقطناه على المنقح سنتوصل إلى القيم التالية :

G = 878977914 Y = 972424840 P = 4150976723 a = أول 8 أحرف من السريال b = ثاني 8 أحرف من السريال M = الاسم	- طبعاً (G, Y, P) : بالنظام العشري ويجب تحويلها إلى الصنف TFGInt . - (a, b) : بالنظام الستة عشري (حقل السريال لا يقبل إلا نظام hex) , ويجب تحويلها إلى القاعدة 256 لاستخدامها في الإجراء ElGamalSign (راجع تعليمات هذا الإجراء وستلاحظ التحويل في البداية إلى 256 إلى الصنف TFGInt).

- إذن السريال الصحيح يمثل جمع القيمتين (a , b) ولكن كيف يمكن الحصول على هذه القيم من خلال الاسم ؟
هنا يجب علينا استخدام الإجراء ElGamalSign الذي ينتج لنا القيمتين (a,b) الصحيحتين للاسم.
لنستعرض هذا الإجراء ولنشاهد ما ينقصنا :

```
Procedure ElGamalSign (M : String; Var p, g, x, k : TFGInt; Var a, b : String);
```

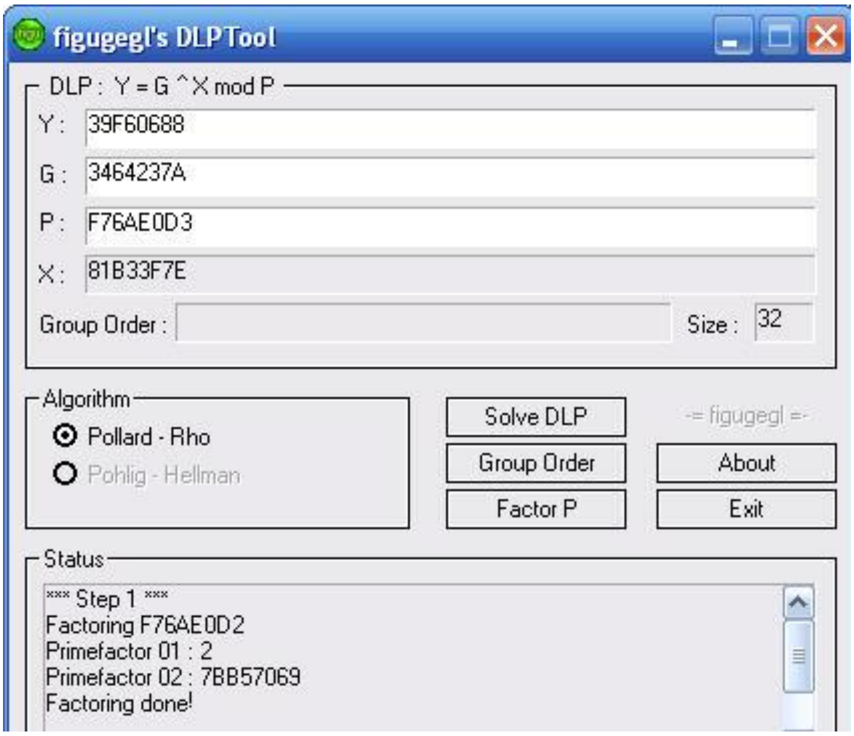
كل القيم متوفرة لدينا باستثناء القيمة X ولإيجاد هذه القيمة لديك إحدى الطريقتين التاليتين :

6.1. الطريقة الأولى : استخدام برنامج DLPTool

هذا البرنامج يحسب لنا قيمة X ولكنه يطلب القيم بالنظام Hex وبتحويل الأرقام إلى Hex نجد :

G = 3464237A , Y = 39F60688 , P = F76AE0D3

ندخل هذه القيم كما في الصورة ثم نضغط Solve DLP والذي يقوم بتطبيق المعادلة : **Y = G ^ X Mod P**

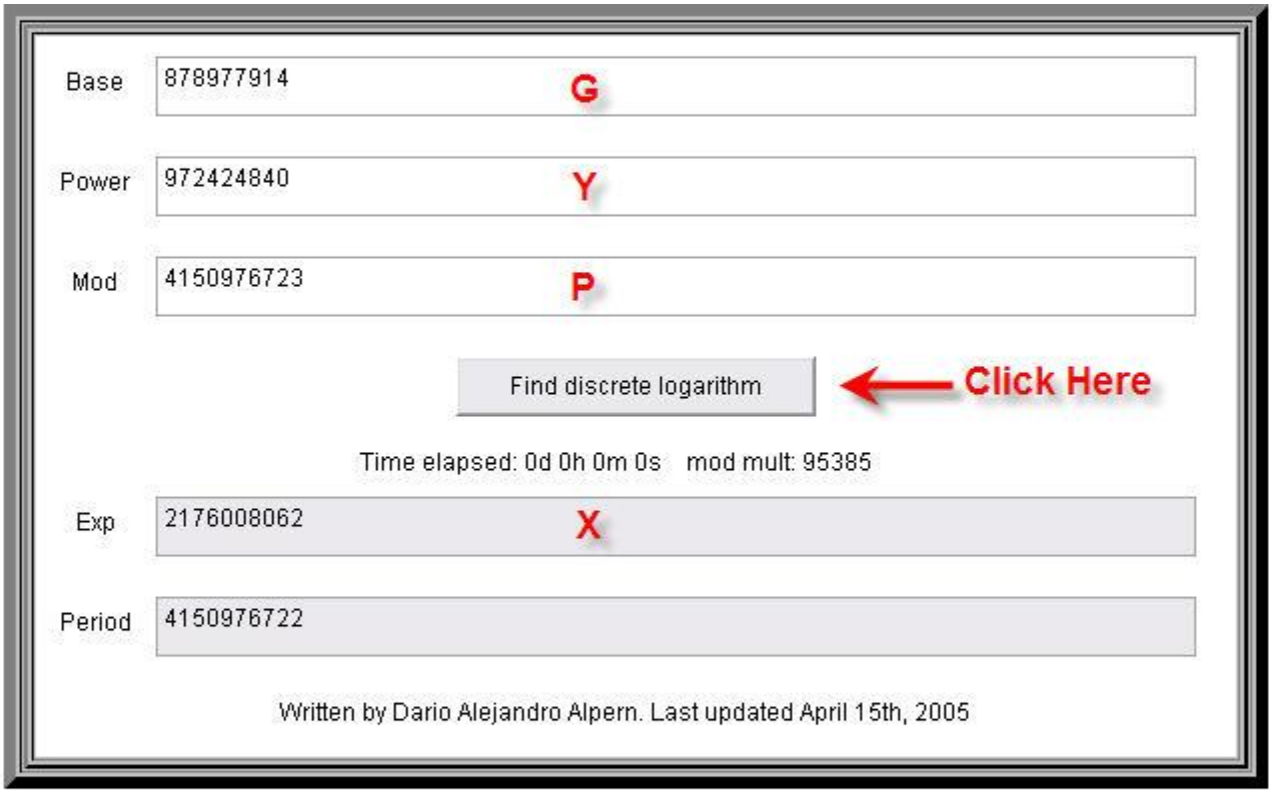


و هكذا نحصل على قيمة : **X = 81B33F7E (Hex) = 2176008062 (Decimal)**

6.2. الطريقة الثانية : استخدام الموقع



والتطبيق في هذا الموقع يقبل القيم بالنظام العشري :



طبعاً المدة التي يستغرقها حل المعادلة تختلف حسب طول المفتاح المستخدم، وقد استخدمنا في هذا التمرين مفتاحاً صغيراً نسبياً بطول 32 Bits بهدف التعليم، والأمر سيكون أصعب بكثير في حال 1024 bits .

7. سادساً : كتابة الكيجن

سنستخدم هنا لغة Delphi 7 وسنكتب الشيفرة الخاصة بتحصيل السريال (السورس كود مرفق) :

```
procedure TForm1.GetSerialClick(Sender: TObject);
Var
FGInt_k, FGint_g,FGint_p,FGInt_x : TFGInt ;
str256_a,str256_b,strhex_a,strhex_b, g,p,X : String ;
begin
   If Length ( Tname.Text) < 1 then
   begin
     Tpass.text:= 'Name > 0';                  { الاسم لا يجوز أن يكون فارغ }
      Exit;
    end;
```

```
g:='878977914';
Base10StringToFGInt(g,FGInt_g);
p:='4150976723';
Base10StringToFGInt(p,FGInt_p);
X:='2176008062';                          { إعداد الثوابت وتحويلها إلى FGInt }
Base10StringToFGInt(X ,FGInt_x);
Base10StringToFGInt('1',FGInt_k);

ElGamalSign(tname.text,FGInt_p,FGInt_g,FGInt_x,FGInt_k,str256_a,str256_b);
ConvertBase256StringToHexString(str256_a,strhex_a);
ConvertBase256StringToHexString(str256_b,strhex_b);
Tpass.text:=strhex_a + strhex_b;          {السريال النهائي a+b }
end;
```

8. كلمة ختامية

- تم التطرق هنا إلى التحقق من التواقيع الرقمية الخاصة بخوارزمية ElGamal وهناك مهمتين إضافيتين تقوم بهما هذه الخوارزمية، وهما: التشفير وفك التشفير، وسيكون الأمر سهلاً عليك إن فهمت الشرح السابق .
- لا يكفي هذا الشرح وحده لإيصالك إلى مستوىً متقدمٍ. بل ينبغي عليك الاطلاع والقراءة لدراسة المزيد من الأساليب والإجرائيات المتبعة في التشفير وفك التشفير وملحقاتها.

استغلال خوارزمية RSA في مكتبة RSACrypt

Keygenning an RSA Target (1)

1. مقدمة وتعاريف

1.1 لمحة عن خوارزمية RSA

إن خوارزمية RSA تعد واحدة من أكثر خوارزميات التشفير استعمالاً في مجال الحاسبات، وهي عبارة عن Public_key_cipher صممت في معهد تكنولوجيا ماساتشوستس MIT الشهير بمدينة كامبريدج عام 1978.

الاسم مقتبس من الأحرف الأولى لمصممي الخوارزمية الثلاث: Ron Rivest , Adi Shamir & Les Andleman

منذ تاريخ 2000/9/20 أصبحت هذه الخوارزمية متاحة للجميع مجاناً، و لم تعد حكراً على حكومة الولايات المتحدة.

1.2 التشفير غير المتناظر

يعتمد التشفير غير المتناظر على مفتاحين لا يشبه أحدهما الآخر ويطلق عليهما اسم Public key وPrivate key حيث أن المفتاح العام متاح للجميع وبه يتم تشفير البيانات، وأما المفتاح الخاص فيتم به فك تشفير تلك البيانات.

إذن من عمل هذا الأخير يتبين لنا أنه يجب علينا إخفاؤه، ومن اسمه كذلك تظهر أهميته.

ويسمى الـ Public key بـ N وبه يتم تشفير البيانات.

ويسمى الـ Private key بـ D وبه يتم فك تشفير البيانات.

2. مثال تطبيقي

سنشرح عملهما بمثالٍ يقربهما للأذهان:

قام كريم بتشفير معلومات بالمفتاح N ثم أرسلها إلى حمزة. ثم قام حمزة بفك تشفيرها وذلك بالمفتاح D؛ حيث أن تلك المعلومات لا يمكن قراءتها إلا من قبل حمزة.

هذا ما يحدث في البرامج التي تعتمد في حمايتها على هذه الخوارزميات. وتعتبر كذلك من بين الخوارزميات الأكثر صعوبة في الكسر.

في مثالنا الذي نحن بصدد شرحه سنستعمل مكتبة الـــ RSACrypt التي سنستخدم منها الدالة RSAencrypt وأما الدالة RSAdecrypt فهي المستعملة في البرنامج.

عملهما:

- RSAencrypt :

لهذه الدالة أربع بارامترات، وعملها هو كالتالي :

$$C = M^{E} \bmod N$$

حيث أن N هو المفتاح العام، وE هو الأس وقيمته هي 65537 بالديسمال و10001 بالهكس، وM هي الرسالة التي نريد تشفيرها، وأما C فهي الرسالة بعد التشفير .

- RSAdecrypt :

لهذه الدالة كذالك أربع بارامترات، ومبدأها في فك التشفير هو كالتالي :

$$M = C^{D} \bmod N$$

حيث أن N هو المفتاح العام، وD هو المفتاح الخاص ويستعمل مكان الأس E ، وC هي الرسالة المشفرة، وM هي الرسالة بعد فك التشفير.

إذن: الدالة الأولى هي الموجودة في مثالنا هذا، وهذا بديهي لأنها تستعمل قيمتين معروفتين وهما N و E ، وأما الدالة الثانية فهي بيت القصيد حيث أنها تستعمل القيم N وD.

بالنسبة لــ N فنحن نعرفها وأما D فلا نعلم عنها شيئاً. وللحصول على القيمة D فإننا نقوم بعملية Factoring أي التحليل إلى عوامل، وذلك بالأداة Msieve أو الأداة RSA Tool.

سنبدأ بتطبيقي عملي وسيكون ضحيتنا البرنامج التجاري

Amor SWF to Video Converter v2.9.9.5



- PEiD أو AT4RE FastScanner.
- OllyDbg.
- QuickUnpack.
- Msieve.
- RSA Tool.

عند فتحك لمربع الحوار الخاص بالتسجيل سيطلب منك في القسم الأول كتابة اسمك، وفي القسم الثاني كتابة تشفير لاسمك؛ حيث أنه سيقوم بفك التشفير المدخل ومقارنته مع الاسم المدخل، وذلك بمساعدة المكتبة RSACrypt، وهذه الأخيرة هي بيت القصيد؛ حيث أنك إذا أردت فك تشفير نص ما فإنك تستعين بـــ :

- الأس 63357 بالديسمال، 10001 بالهكس.
- المفتاح N.

وأما إذا أردت تشفير النص فإنك تستعين بـــ :

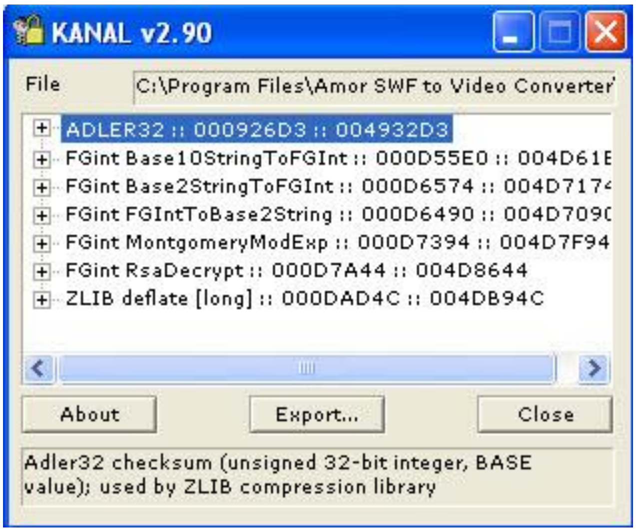
- المفتاح N.
- المفتاح D.

بالنسبة للأس فنحن نعرفه، والمفتاح N كذلك نعرفه، ولكن المفتاح D لا نعرف عنه شيئا فهو مجهول!!

إذن ما الحل ؟ الحل : هو عملية Factoring للمفتاح N.

بداية دعنا نقوم بفك ضغط البرنامج بالبرنامج الشهير QuickUnpack؛ لأن البرنامج الضحية مضغوط كما أوضح لنا برنامج الفحص PEiD.

بعد ذلك لنقم بفحصه مرة أخرى ببرنامج PEiD لنعرف أي مكتبات التشفير يستعمل وذلك بواسطة الأداة المساعدة Krypto ANALyzer المضافة لبرنامج الفحص PEiD، والنتيجة كما تلاحظ : إنه يستعمل المكتبة FGint.



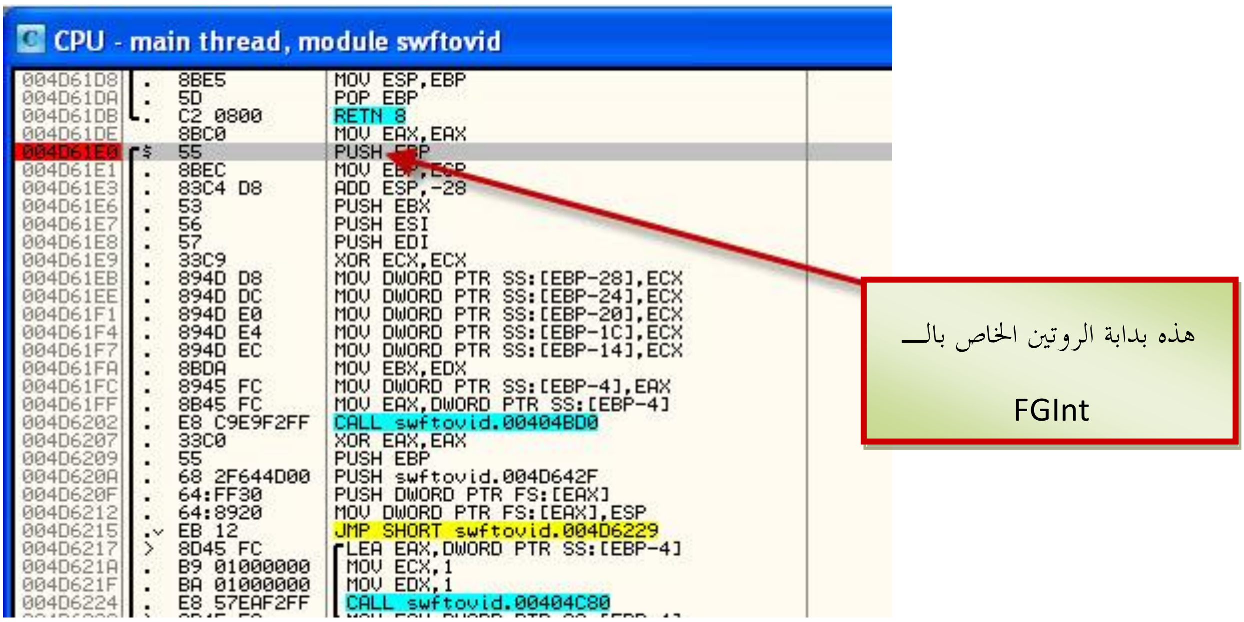
سنقوم بنسخ ما نراه ولصقه في المفكرة، وذلك بواسطة الزر Export الذي تراه في الصورة السابقة بهذا النحو:

Export... ===> To Clipboard ===> list of items with references

الدالة الأولى Base10StringToFgint : تقوم بالتحويل من الديسيمال إلى الـــ FGint.

وأما الدالة الثانية والثالثة والرابعة فلا أهمية لهم عندنا في هدا الدرس، فنحن نريد أن نعطيك الخيط وأنت تكمل المشوار، فإن تعلمت شيئاً جديداً فلا تبخل علينا به.

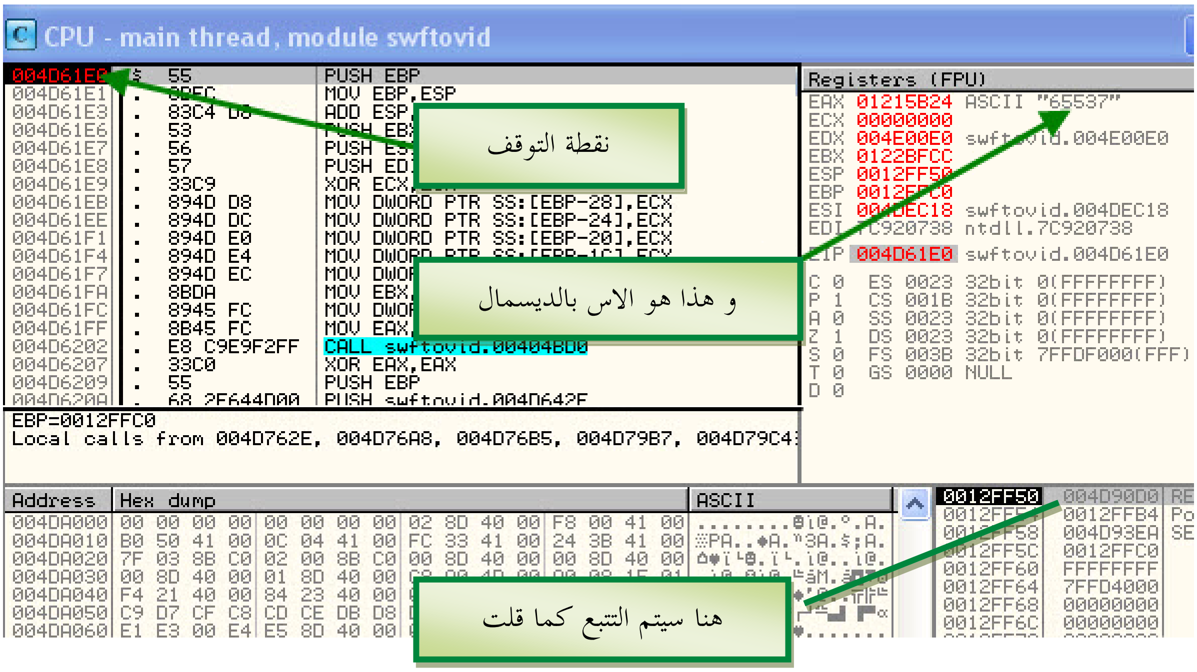
لننسخ العنوان 4D61E0 والعنوان 4D8644 ثم نتتبعهما في OllyDbg الذي يقودنا إلى بداية الدالة Base10StringToFgint في العنوان الأول كما يظهر في الصورة التي أمامك :



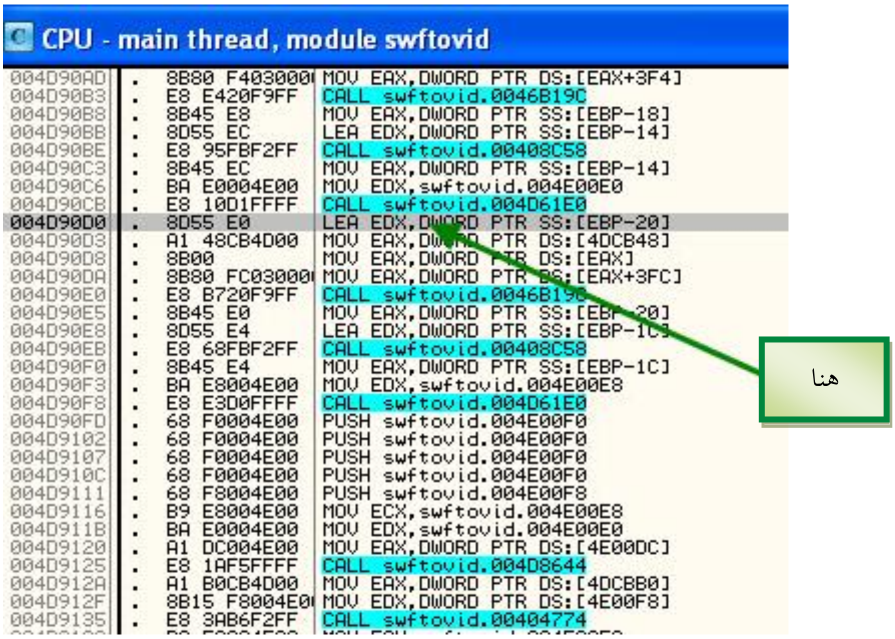
ولكن هذا ليس كل شيء؛ فإن برنامجنا لن يتوقف عند نقطة التوقف؛ لأنه يقرأ معلومات التسجيل التي أدخلتها سلفاً، وذلك أثناء تنفيذ البرنامج، وهذه المعلومات حفظها البرنامج في ملف swftovideo.ini.

إذن: يجب ملء معلومات التسجيل لإنشاء الملف كي يتسنَّ لك التوقف عند النقطة الظاهرة في الصورة.

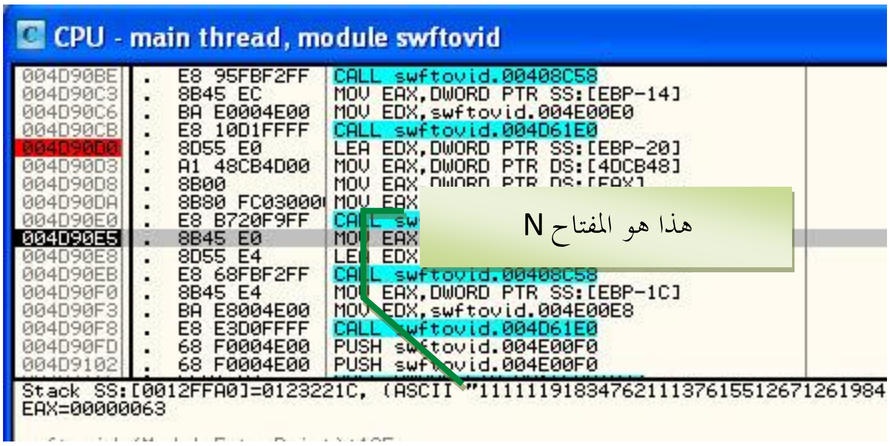
حسنا ستتوقف عند تلك النقطة لنشاهد ونعرف من أي عنوان تم استدعاء هذا الروتين؟



سنذهب إلى الـ Stack ثم بالزر الأيمن للفأرة نختار البند Follow in Disassembler ، وبعد الضغط على هذا الخيار سنكون هنا كما هو واضح بالصورة :



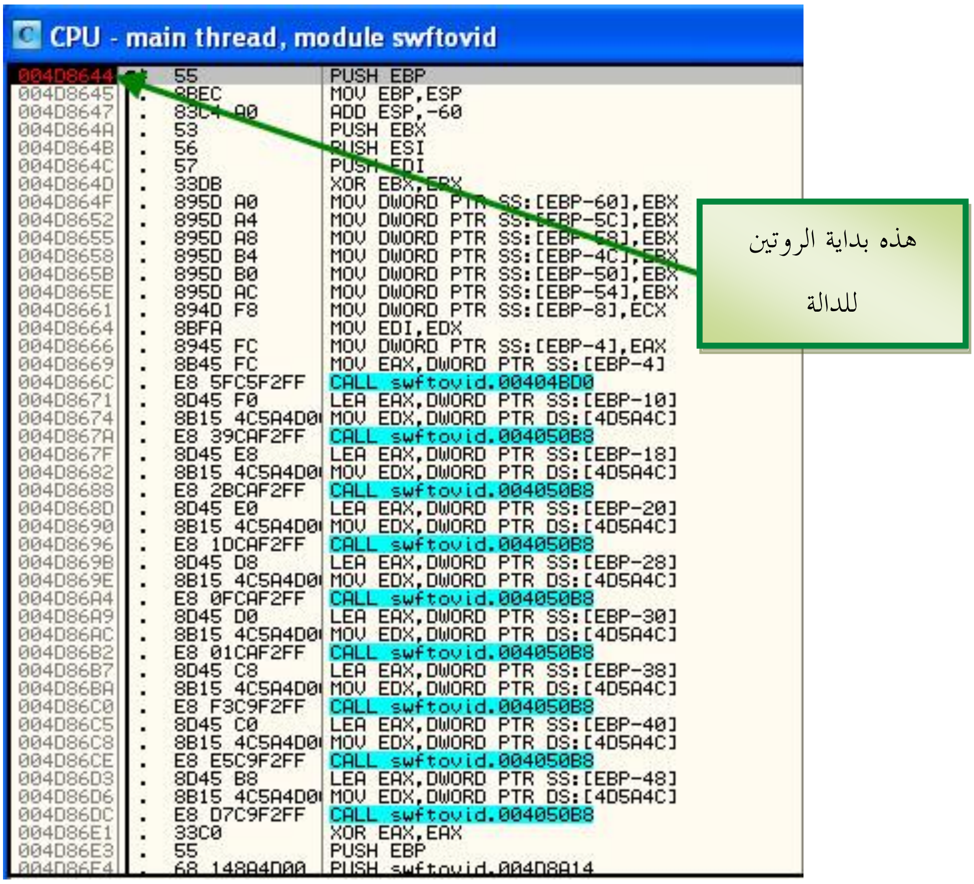
لنضع نقطة توقف عند العنوان 4D90D0 كي نخرج من الروتين الذي أخذنا فكرة عن ماهية عمله، ثم بعد ذلك نقوم بالتتبع إلى أن نصل إلى العنوان 4D90E5 الذي سيظهر عنده المفتاح N كما في الصورة :



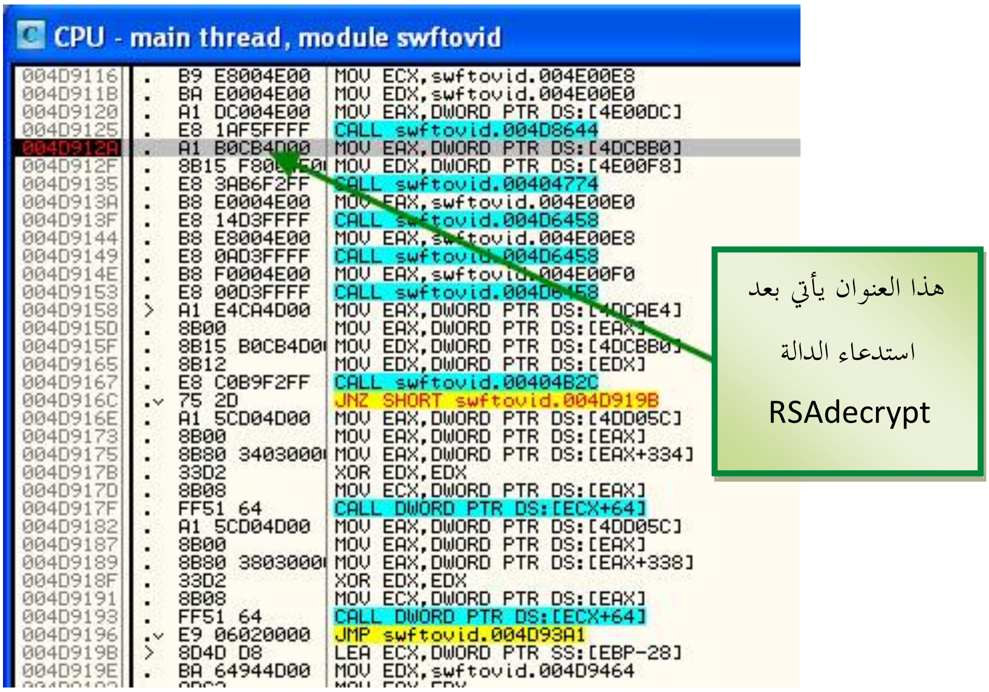
وإذا تتبعنا قليلاً سنجد نداءً آخر يستدعي الدالة Base10StringToFgint.

إذن: نستخلص مما سبق بأن المفتاح N سيحول كذلك إلى الـــ FGint. يعني يتم تجهيز المفتاحين لعملية فك التشفير.

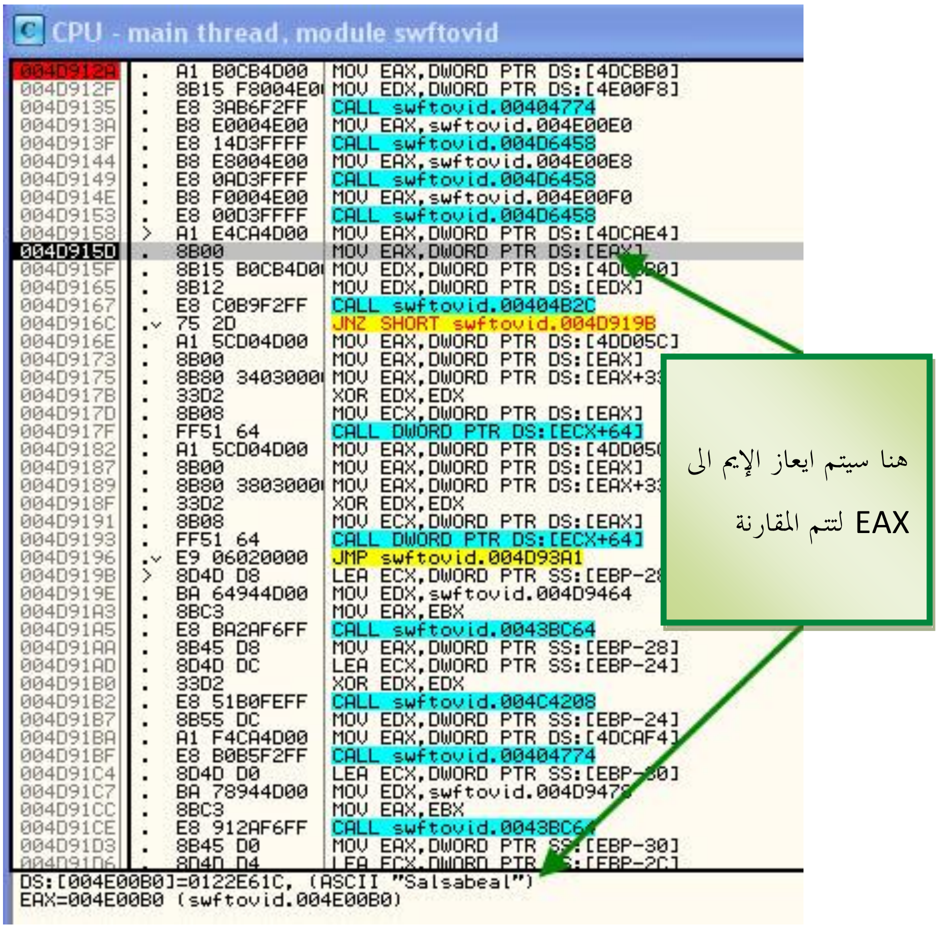
حسناً لنكمل... بعد التتبع سنجد أنفسنا عند نقطة التوقف الخاصة بالدالة RSAdecrypt كما تشاهد في الصورة :



سنفعل كما فعلنا سابقاً؛ حيث سنقوم بالرجوع إلى النداء الذي يستدعي هذه الدالة وذلك بواسطة الـــ Stack ، وبعدها سنكون هنا :



إذن: هنا يتم فك التشفير، ثم تأتي بعد ذلك عملية المقارنة أي مقارنة الاسم المدخل مع التشفير الذي تم فكه. شاهد الصورة التي توضح ذلك :



كما هو ظاهر في الصورة فهنالك استدعاء للمقارنة بعدها تأتي قفزة شرطية، أي اقفز إذا لم تتحقق المقارنة ولا تقفز إذا تحققت المقارنة.

إذن ماذا سنفعل؟

من خلال ما ذكرناه سابقاً عن RSAdecrypt وRSAencrypt سنستعمل الدالة الثانية، وهي RSAencrypt التي تقوم بتشفير الاسم، مع العلم أنها تستخدم المفتاح D وهذا المفتاح مجهول بالنسبة إلينا! وهنا يأتي دور الأداة Msieve.

ستقول : لماذا هذه الأداة Msieve ؟ وما فائدة الأداة الأخرى RSA Tool ؟

سنقول لك : إن الأداة التي سوف نستعملها ستسرع قليلاً عملية الـــ Factoring وستعطينا الرقمين p و q ليتسنَّ لنا حساب المفتاح D عبر الأداة RSA Tool.

نشكر مبرمج هذه الأداة المميزة غير أن فيها عيباً واحداً، وهو أنها لا تستقبل إلا الأعداد العشرية.

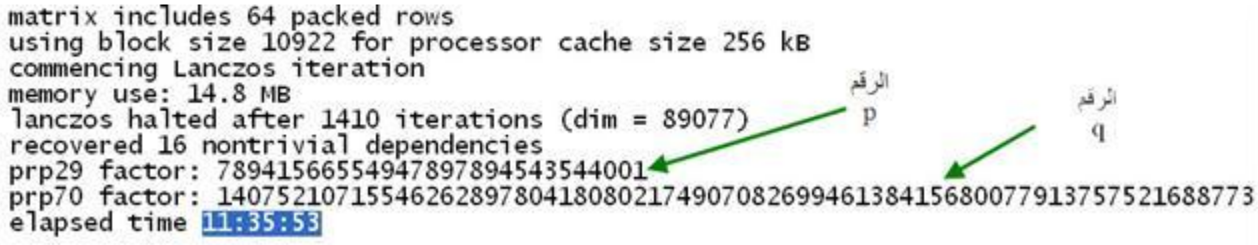
من أراد المزيد عن كيفية استعمال هذه الأداة فالأخ FoxZom تكفل بذلك وبشرحٍ فلاشي كذلك.

نفتح الأداة Msieve ثم ننسخ فيها المفتاح N الذي حصلنا عليه أثناء التنقيح، وهو:

N= 111111918347621113761551267126198454426053267642844509165651820825173537473218924746426621953200773

استغرق الجهاز الذي نعمل عليه في استخراج الرقمين p و q حوالي 11:35:53.

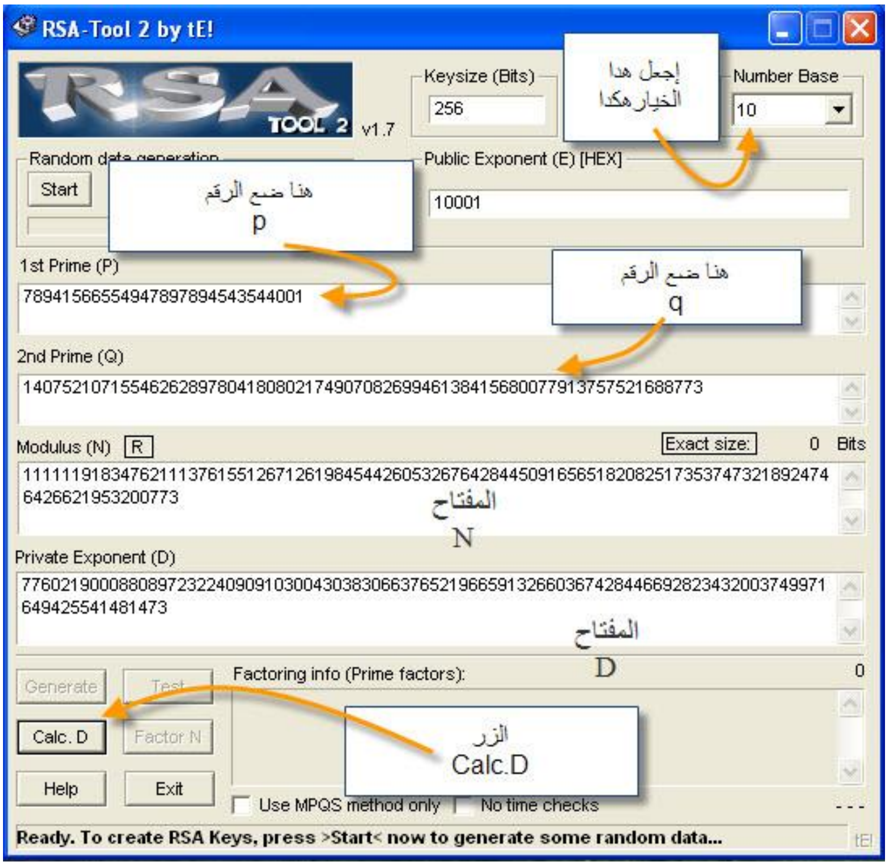
إلى الآن لدينا الرقمين p و q كما يظهر في الـــ LogFile :



```
matrix includes 64 packed rows
using block size 10922 for processor cache size 256 kB
commencing Lanczos iteration
memory use: 14.8 MB
lanczos halted after 1410 iterations (dim = 89077)
recovered 16 nontrivial dependencies
prp29 factor: 78941566554947897894543544001
prp70 factor: 1407521071554626289780418080217490708269946138415680077913757521688773
elapsed time 11:35:53
```


والآن كيف سنستخرج المفتاح D ؟ الجواب : عبر الأداة RSA Tool.

شغل الأداة ثم قم بنسخ الرقمين p و q وبعدها اضغط على الزر Calc.D كما يظهر في الصورة :



سنشرح لك ما تم فعله :

أولاً: حددنا القاعدة التي سنتعامل معها، وذلك بالذهاب إلى خيار الـــ Number Base واختيار10؛ لأننا نتعامل مع أرقام بالنظام العشري.

ثانياً: نسخنا كلاً من الرقمين p و q في المكان المخصص لهما، ثم ضغطنا على الزر Calc.D لحساب المفتاح D وهذا كل شيء.

إذن لدينا N و D ، وماذا بعد؟ الآن ستأتي برمجة الكيجن، وسنقوم ببرمجته بلغة الدلفي، وسنستعمل لذلك الدالة RSAencypt والمفتاحين N و D كما تشاهد أمامك في الشكل :



```
uses FGInt, FGIntRSA;

{$R *.dfm}
resourcestring
e1 = '65537';

n1= '1111119183476211137615512671261984544260532676428445091656518208251735374
d1= '7760219000880897232240909103004303830663765219665913266036742844669282343
procedure TForm1.Button1Click(Sender: TObject);
var
   test, b64:string;
   e,n:tfgint;
begin
   test:=edit1.text;
   Base10StringToFGInt(d1, e);
   Base10StringToFGInt(n1, n);
   RSAEncrypt(test, e, n, test);
   ConvertBase256to64(test,b64);
   edit2.Text:=  ( b64);
   FGIntDestroy(e);
   FGIntDestroy(n);
end;
```


كما يظهر في الصورة N1 هي N و D1 هي D ، والدالة Base10StringToFGint كما ذكرنا آنفاً تقوم بتحويل الأعداد العشرية إلى الـــ FGint والدالة RSAencrypt تقوم بهذه العملية : C = M^D mod N

- M هو test وكذلك هو C وبما أن القيمة ستتغير للمتغير test بعد الدالة RSAencrypt فليس هناك أي مشكلة في تكراره.
- e هو d1 بعد دالة التحويل إلى الـــ FGint .
- n هي n1 بعد دالة التحويل إلى FGint .
- C هي test كما يظهر في الصورة .

لنقم بتجربة البرنامج :



وبهذا نكون قد انتهينا من شرح أكثر الخوارزميات استخداماً وشيوعاً، ومع مرور الوقت ستجد أنك قادر على فهم هذه الخوارزمية أكثر وأكثر..

استغلال خوارزمية الـRSA في مكتبة FGIntRSA

Keygenning an RSA Target (2)

1. المقدمة

سيتم تطبيق هذا الدرس على برامج شركة Absolute وبالتحديد برنامج Absolute sound recorder

إن كتابة كيجن لهذا البرنامج غاية في السهولة خاصة إذا عرفنا كيفية التعامل مع البرامج التي تستخدم خوارزميات التشفير وهذا ما سنقوم بتوضيحه إن شاء الله.

2. فحص البرنامج

إن أهم خطوة يجب أن تقوم بها بداية للتعامل مع أي هدف تريد كسره هو دراسته قبل بدء التعقب (Tracing) بـــ F8 وF7 وهذه الخطوة ورغم أهميتها البالغة يغفلها كثير من الناس.

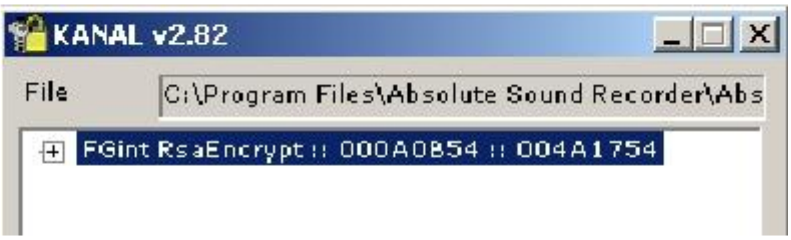
وهذه المرحلة تتم عبر ثلاث أدوات، هي: PEeditor و PEID و IDA

هذه المرحلة نقوم بها دائماً علماً أنها غير إلزامية، ولكنها غالباً ما توفر عليك الكثير من الجهد والوقت.

- الأداة الأولى PEeditor : ومن خلالها يمكنك معرفة هل الملف مشفر أم لا ؟ وفي العديد من الحالات يمكنك معرفة نوع التشفير حتى قبل برامج الفحص كـــ PEiD أو AT4RE FastScanner كما تمكنك من معرفة الدوال المستوردة الخارجة عن المكتبات المعتادة، وهذا يسهل تحديد المكتبة التي تحتوي خوارزمية التحقق من السريال.

تمكنك هذه المرحلة أيضا من عمل إطلالة على بعض القيم المهمة في الـــ PE Header والتي قد تُحدث لك تمنيحاً في الجهاز، كما هو الحال مع Private exe Protector مثلاً؛ فالسيكشن الكبيرة التي يضيفها تمنح الجهاز أوتوماتيكياً.

- الأداة الثانية PEiD : وهي غنية عن التعريف، لكن ما يهمنا هنا هو الأداة المساعدة Plugin المضافة لها، وهي : Kanal Krypto Analyzer والتي قد تعطيك بعض المعلومات التي تجعلك تتقدم بـــ 80 أو 90 % في كسرك للبرنامج حتى قبل الوصول إلى المنقح، كما هو الحال مع هدفنا الحالي : Absolute sound recorder :



وترجمتها لمن لم يصادف هكذا مثال من قبل: أن البرنامج يستخدم RSA كما تدل على ذلك الكلمة RSAEncrypt أما الكلمة FGint فتعني أن البرنامج يستخدم المكتبة FGint أو FGintRSA للتشفير، وهما مكتبتان مشهورتان عند مبرمجي الدلفي ويمكن الحصول عليهما من جوجل.

تم إرفاق مكتبة الـــ FGint والـــ FGintRSA مع مرفقات الكتاب.

3. كسر البرنامج

- الأداة الثالثة IDA : طبعاً لن نضيع الوقت في بيان كيفية الوصول إلى المكان المهم في خوارزمية التحقق، فأكثرنا يستطيع الوصول إليها بسهولة عن طريق OllyDbg أو حتى Dede .

وهذه صورة البرنامج بعد التحليل الأولي عند النقطة المهمة وبعد تسمية الدوال المهمة :

```
; Attributes: bp-based frame

IMPORTANT_ROUTINE proc near

var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
push    0
push    0
push    ebx
mov     ebx, eax
xor     eax, eax
push    ebp
push    offset sub_4A9F8F
push    dword ptr fs:[eax]
mov     fs:[eax], esp
lea     edx, [ebp+var_4]
mov     eax, [ebx+314h]
call    sub_46E208
lea     edx, [ebp+var_8]
mov     eax, [ebx+318h]
call    sub_46E208
mov     eax, ds:off_4AEBB8
mov     eax, [eax]
mov     ecx, [ebp+var_8]
mov     edx, [ebp+var_4]
```

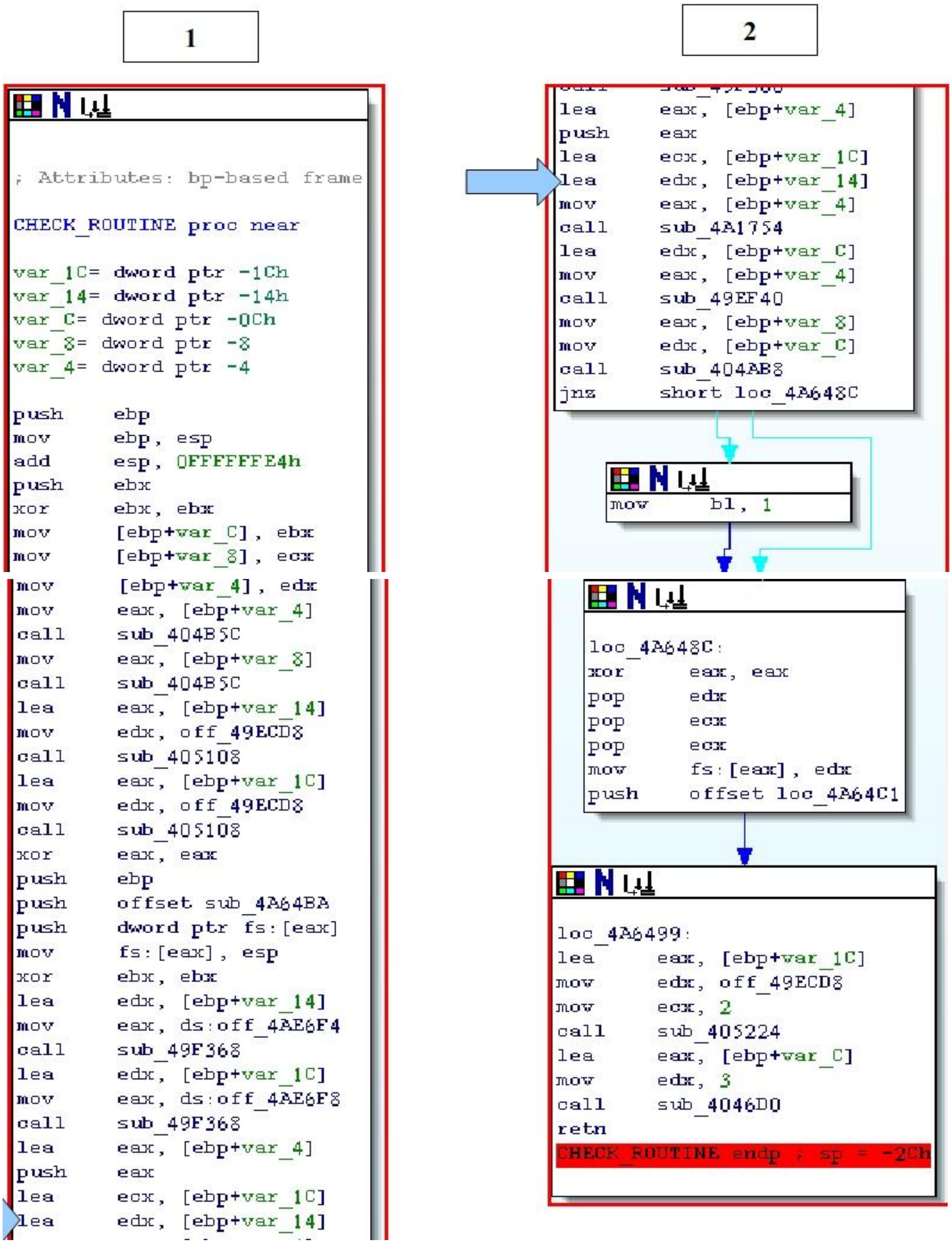
```
call    CHECK_SUBROUTINE ; خوارزمية التحقق بعد اعادة تسميتها
test    al, al           ; هل تحقق الشرط
jz      short BAD_BOY    ; اقفز الى عدم التسجيل اذا لم يتحقق الشرط
```

```
mov     eax, ds:off_4AEBB8
mov     eax, [eax]
mov     edx, [ebp+var_4]
call    sub_4A6770
push    40h
mov     ecx, offset s_Congratulatio ; "Congratulations!"
mov     edx, offset s_RegisterSucce ; "Register successfully!  Thank you for y"...
mov     eax, ds:off_4AEA74
mov     eax, [eax]
call    sub_48F074
mov     eax, ebx
call    sub_48B72C
```

```
BAD_BOY:
xor     eax, eax
```

تلاحظون أن الدالة CHECK_SUBROUTINE هي أهم موقع هنا لذا سندخل إليها. وستلاحظون أنه لا يوجد شيء يمكن استنتاجه من التحليل الاستاتيكي (Static Analysis وهو تحليل الكود دون تعقب)، فهل يتوجب علينا استخدام المنقح؟ الجواب: قطعاً لا؛ لأنه مازال هنالك الكثير من المفاجآت التي يخبأها لنا IDA !!

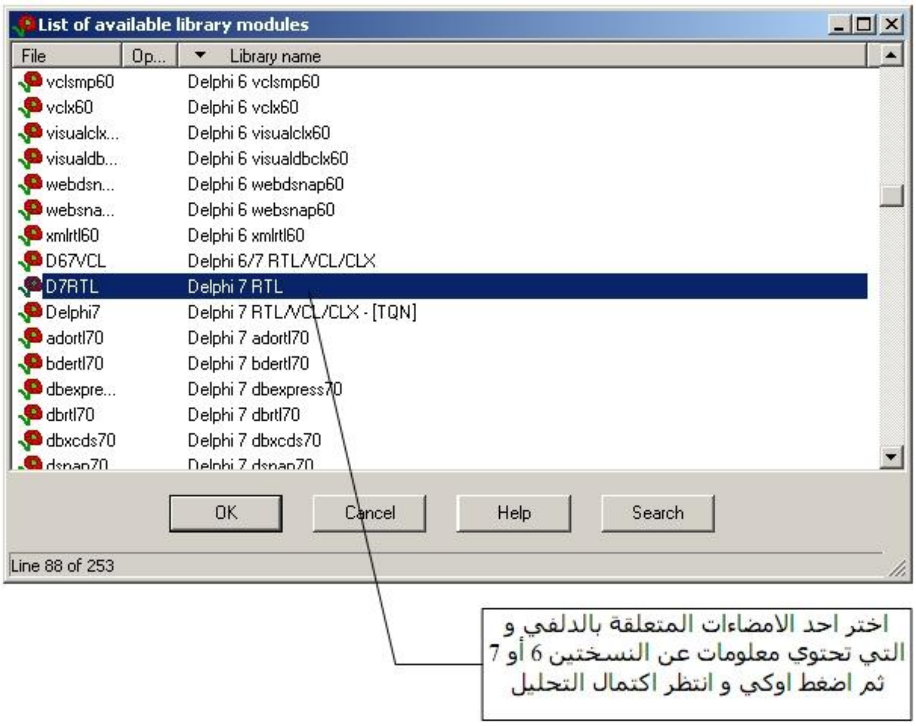
سنستخدم في هذا الدرس خاصية Flirt signatures لكن قبل ذلك هذا سرد لخوارزميات التحقق قبل أي تغيير:



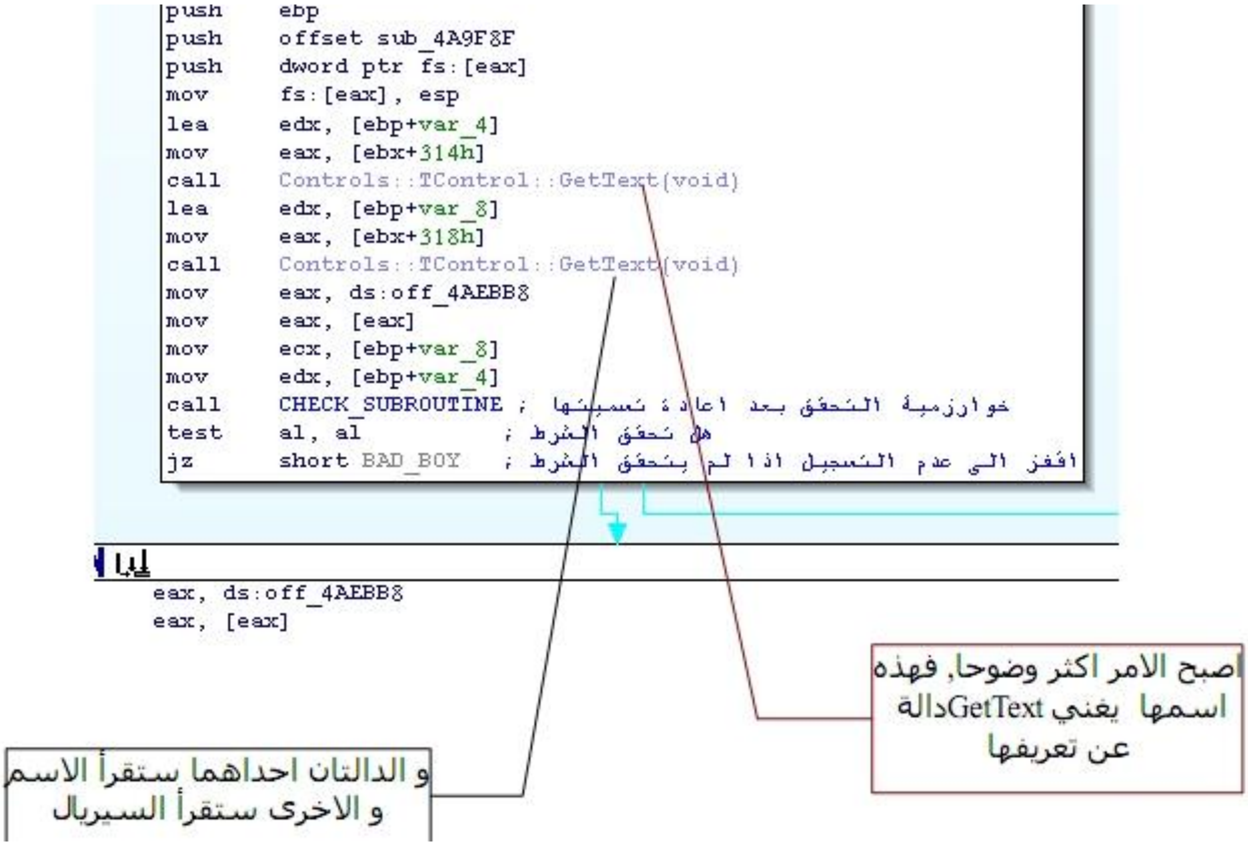
كون البرنامج مبرمج بلغة الدلفي فسنستغل هذه النقطة، والاستغلال يكون كالتالي :



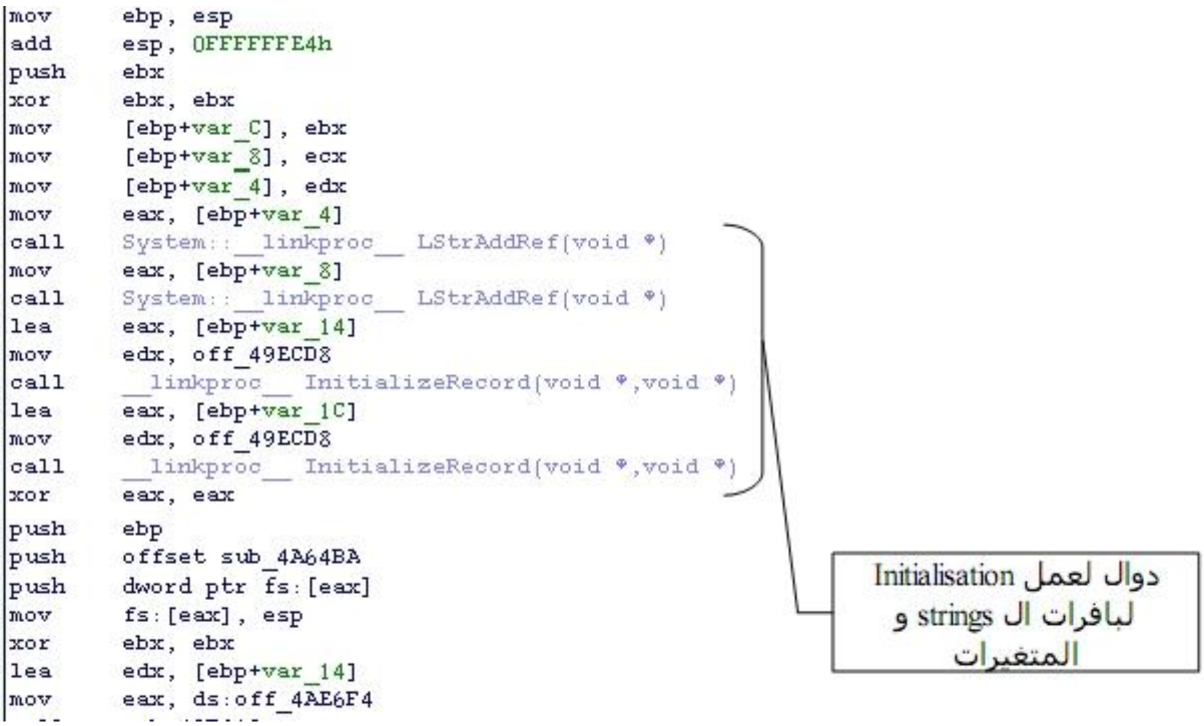
ستخرج لنا النافذة السحرية :



وهذا مظهر الموقع المهم بعد تطبيق الإمضاء :



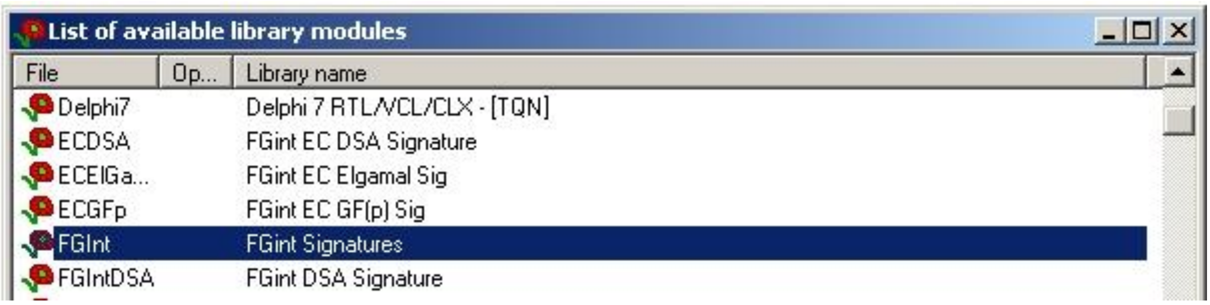
وهذا مظهر خوارزمية التحقق بعد تطبيق التوقيع وقد أصبح أيضا أكثر وضوحاً :



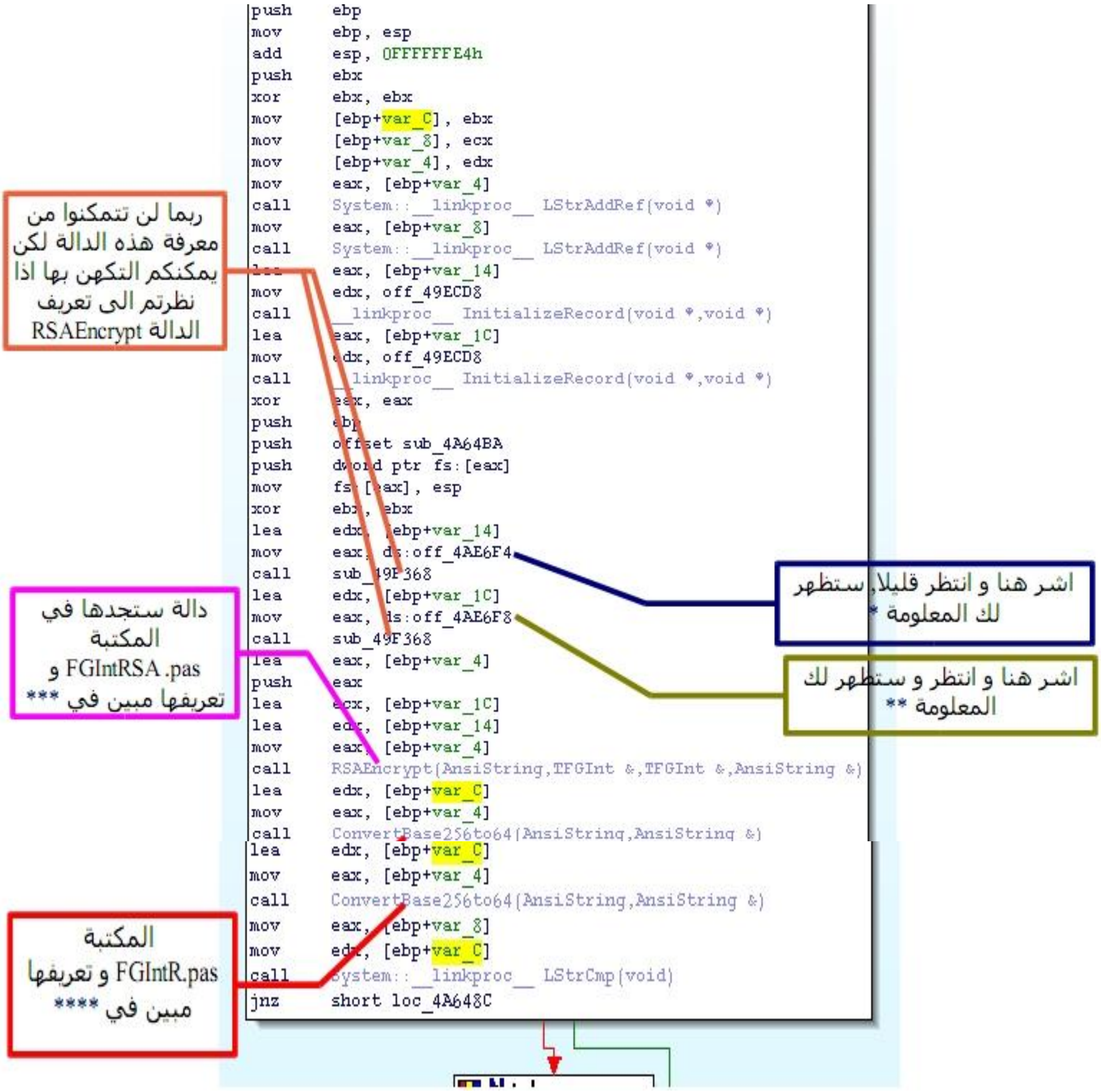
والآن سنستغل نقطة ثانية، وهي أن البرنامج يستخدم المكتبة : FGIntRSA و FGInt والاستغلال بنفس الطريقة لكن هذه المرة باختيار الإمضاءين الموافقين للمكتبتين السابقتين.

> تم إرفاق جميع الإمضاءات المطلوبة في الشرح
>
> مع مرفقات الكتاب.





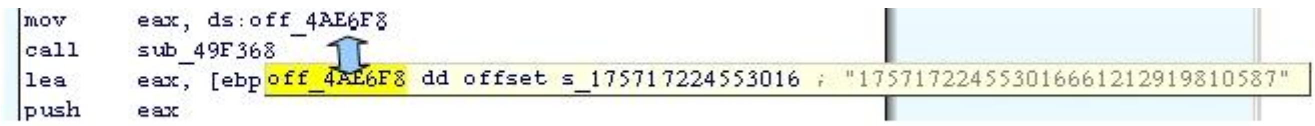
والنتيجة لا تترك أي شك في إمكانية عمل كيجن دون إطلاق المنقح :



:*



:**



:***

```
Procedure RSAEncrypt(P : String; Var exp, modb : TFGInt; Var E : String);
```

حيث :

- P هو النص المراد تشفيره بالـ RSA (في حالتنا هو : الاسم).
- Esp هو الأس (Exponent) (في حالتنا هو : 65537).
- Modb هو باقي القسمة (وفي حالتنا هو : 1757172245530166612129198 10587).
- E هو ذاكرة التخزين المؤقت (Buffer) الذي سيتلقى الناتج المُشفر: (في حالتنا هو الــ Buffer الذي سيتلقى السريال).

وكما تعلمون فإن التشفير بـــ RSA يحقق المعادلة :

$$P^{exp} \bmod modb = E$$

السريال = الاسم ^ 65537 مود175717224553016661212919810587

بقليل من التحليل سيتبين لك أن الدالة sub_49F368 هي للتحويل من String إلى عدد كبير جداً (TFGInt) وهي معرفة في FGInt.pas كالتالي :

```
Procedure Base10StringToFGInt(Base10 : String; Var FGInt : TFGInt);
```

بحيث Base10 هو string ممثل لعدد بالنظام العشري وFGInt هو الــ Buffer الذي سيتلقى عدداً كبيراً جداً موافقاً له.

:****

```
Procedure ConvertBase256to64(Const str256 : String; Var str64 : String);
```

حيث :

- Str256 هو الــ string المراد تحويله (في حالتنا هو السريال الناتج عن RSAEncrypt).
- Str64 هو ذاكرة التخزين المؤقت (Buffer) الذي سيتلقى الــ string المحول.

وهذه قطعة الكود الذي ينتج السريال الصحيح (الكيجن) :

```
procedure TForm1.Gen(Sender: TObject; var Key: Word; Shift: TShiftState);
var
  P , E , mySerial: string;
  Modb, Exp : TFGInt;

begin
  P := edName.Text;
  Base10StringToFGInt('65537', Exp);
```

```
  Base10StringToFGInt('17571722455301666121291981O587', Modb);
  RSAEncrypt(P, Exp, Modb, E);
  ConvertBase256to64(E, mySerial);
  edSerial.Text := mySerial;
end;
```

9. المرفقات

الكيجن والسورس كود مرفق مع مرفقات الكتاب.

الباب الرابع: أفكار وحلول برمجية في الهندسة العكسية

طريقة برمجة Patcher بالأسمبلي

Coding a patcher in A

...ble Extractor 1.34

من هنا تستطيع تحميل المرفقات.

I. مقدمة

تم تطبيق الدرس على برنامج X-Netstat 5.0 ويمكن تطبيقه على أي برنامج آخر.

2. مراحل الدرس

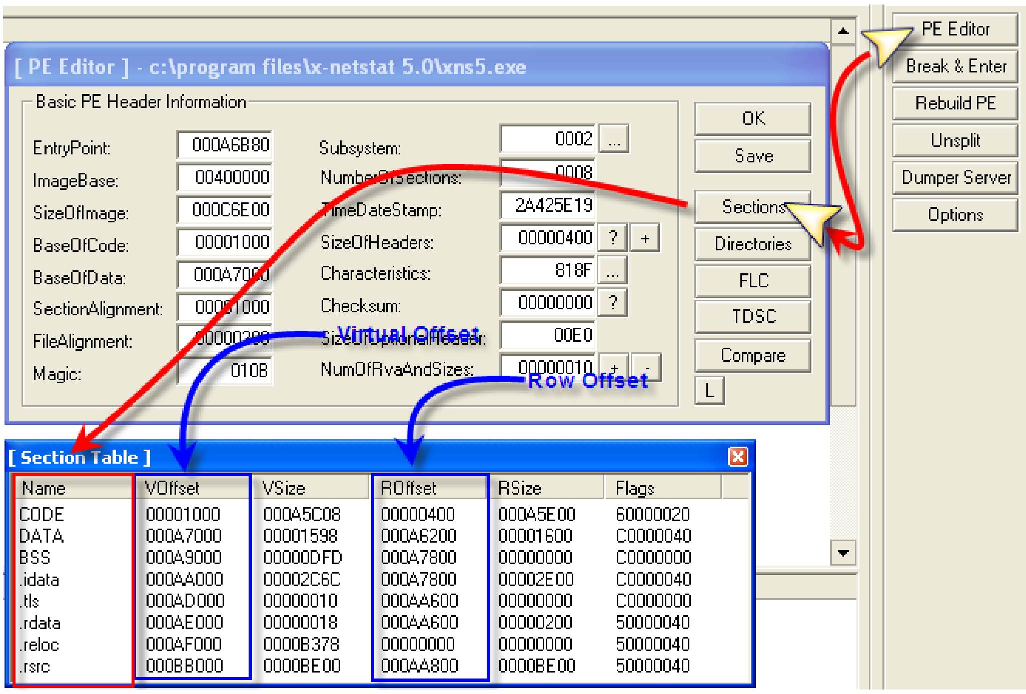
- حساب offset .
- الحصول على قالب وتعديله بواسطة RadAsm .
- التعرف على الدوال الأساسية المستعملة في برمجة Patch .
- الشروع في برمجة الــ Patch .
- تجربة الـــ Patch .
- إضافة تحسينات للــ Patch .
- نصائح وملاحظات.

2.1 حساب OFFSET

يمكن حساب offset باعتماد المعادلة التالية :

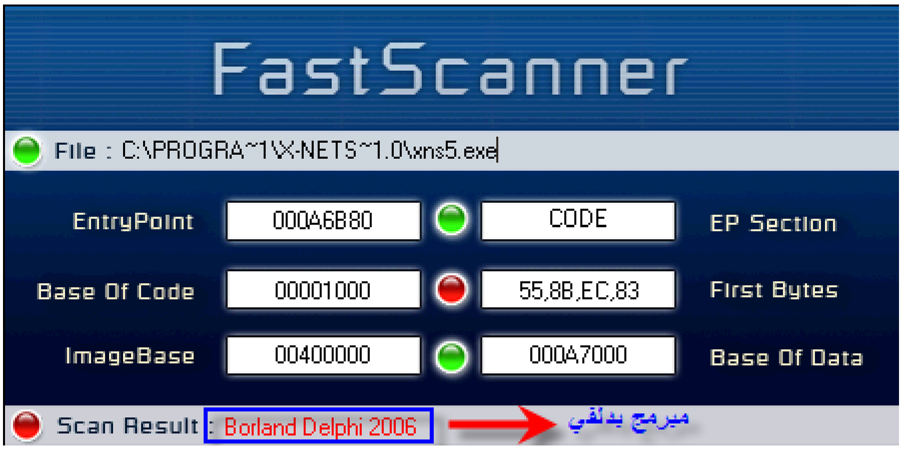
```
Offset = RVA – Virtual Offset + Row Offset
RVA = VA – ImageBase
```

بعد تنصيب برنامج X-Netstat حمله إلى برنامج LordPE ثم انظر الأقسام وذلك بالضغط على الزر Sections للحصول على أقسام البرنامج.



لا تقلق إن لم تفهم فسيأتي المثال فيما بعد.

من أسماء الأقسام نتوقع أن يكون مبرمجً بلغة الدلفي, دعنا نفحصه :

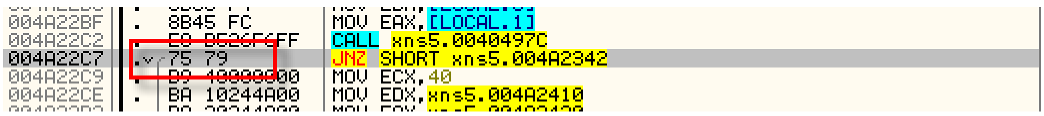


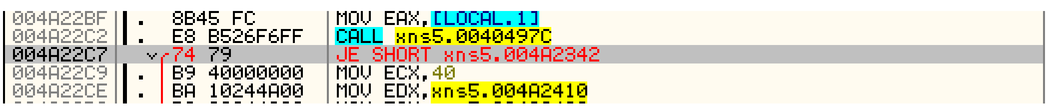
الآن قم بتحميل البرنامج الضحية إلى المنقح OllyDbg ، ولن نشرح طريقة كسره لأنه ليس موضوع الدرس.

بعد أن وجدت القفزة المراد تغييرها كالتالي :



لاحظ معي هاتين الصورتين للمقارنة قبل التعديل وبعد التعديل :





بعد التغيير نلاحظ أنه تم بتغيير بايت واحدة من 75 إلى 74 عند العنوان 004A22C7h في الذاكرة.

و الآن لنقم بحساب الـ offset الموافق لهذا العنوان باعتماد المعادلة المذكورة في أول الدرس:

```
VA =  004A22C7h
RVA = VA - ImageBase =  004A22C7h - 00400000h = A22C7h
Offset =  RVA - Virtual Offset + Row Offset =  A22C7h - 1000h + 400h =  A16C7h
```

إذن offset المطلوب هو A16C7h, وحين نتحدث عن offset فإننا نتحدث عن وضعية الملف على القرص الصلب وليس في الذاكرة .

يمكن التأكد بواسطة محرر هكس مثل WinHex أو HexWorkshop v5 وذلك بالضغط على Ctrl + G أو من خلال القوائم : Edit → Goto... بالنسبة لبرنامج HexWorkshop :

```
000A16C6 FF75 79B9 4000 0000 BA10 .uy.@.....
000A16D0 244A 00B8 3024 4A00 E8FF $J..0$J...
```

لنحتفظ بالمعلومات الأساسية في هذه الفقرة :

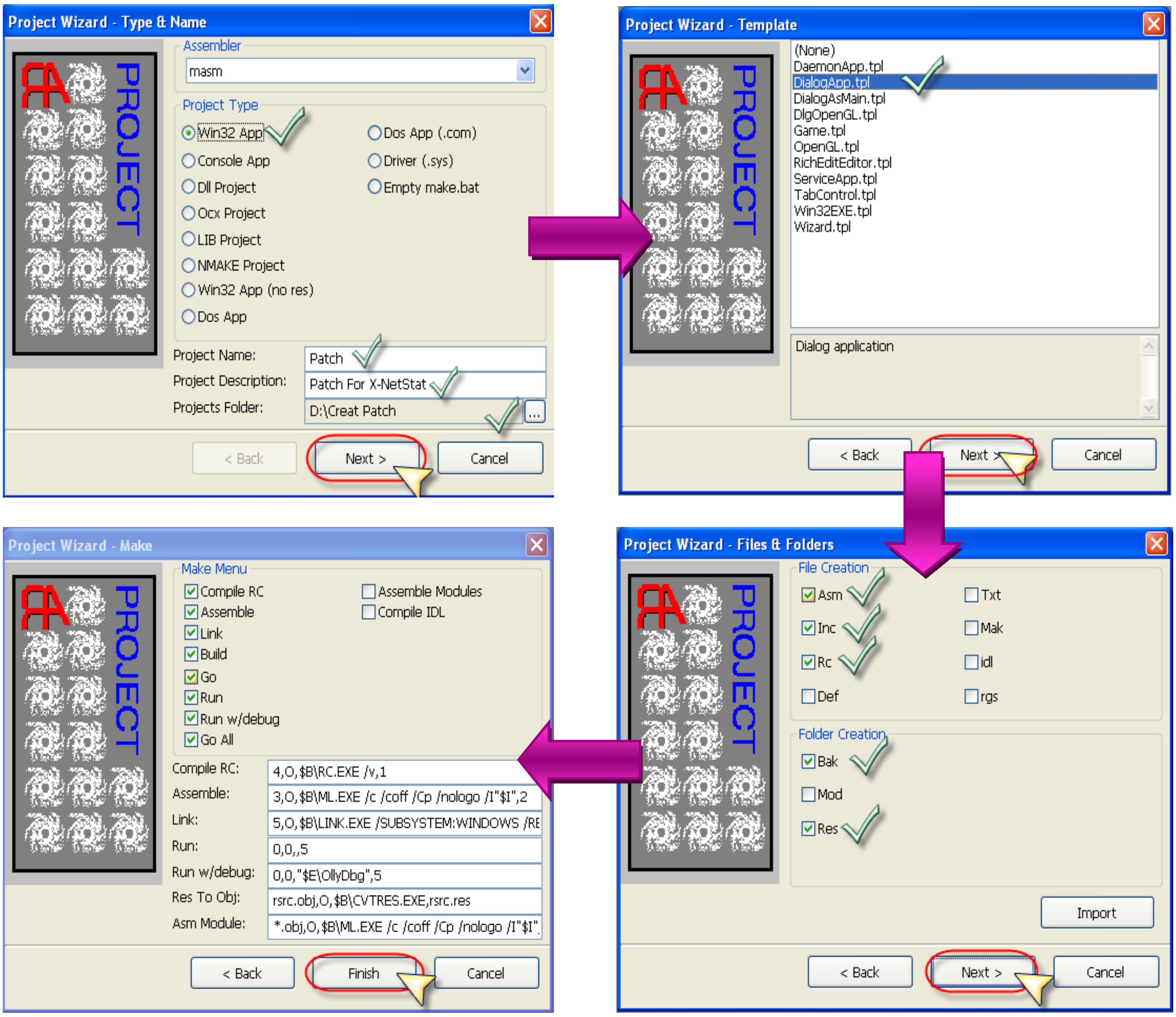
Offset = A16C7h

البايت التي تم تغيرها عند هذا الـــ offset من **75** إلى **74**

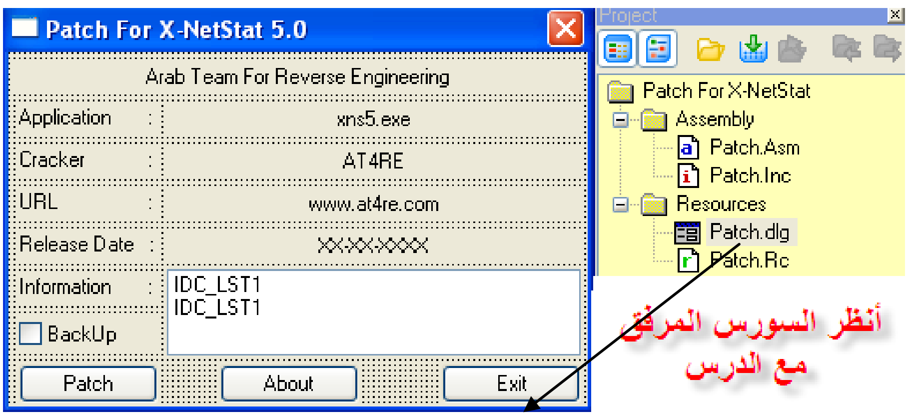
2.2. الحصول على قالب وتعديله بواسطة RadAsm

الآن لنفتح برنامج RadAsm ولنتبع المراحل التالية :

افتح برنامج RadAsm ثم اضغط New Project → File ثم تابع :



بعد ذلك نقوم بتجهيز القالب وليكن كالتالي :



لنعطي الأزرار ID خاصاً، وليكن كالتالي :

ID Pacth = 10 ،ID About = 11 ،ID Exit = 12 ،و ID BackUp=13 و ID List=14

2.3 التعرف على الدوال الأساسية المستعملة في برمجة Patch

HANDLE CreateFile (	دالة إنشاء ملف جديد أو قراءته والحصول على المقبض الخاص بالملف ويدفع هذا الأخير إلى المسجل EAX
LPCTSTR lpFileName	اسم الملف الذي تريد إنشاءه أو قراءته
DWORD dwDesiredAccess	نوعية التعامل مع الملف : للقراءة أو الكتابة أو كلاهما معا
DWORD dwShareMode	لا تزعج نفسك بها واجعلها Write أو Read Write
LPSECURITY_ATTRIBUTES lpSecurityAttributes	0
DWORD dwCreationDistribution	CREATE_NEW لإنشاء ملف جديد وفي حالة كان الملف موجود فلا تفعل شيئاً. CREATE_ALWAYS مثل السابقة إلا أنها تعمل رغم وجود الملف. OPEN_EXISTING فتح الملف – وليس تنفيذه – في حالة وجوده. OPEN_ALWAYS مثل سابقتها لكن باختلاف طفيف. TRUNCATE_EXISTING تمسح معلومات الملف وترجعه إلى حجم منعدم.
DWORD dwFlagsAndAttributes	نوع الملف : مخفي أو عادي وتوجد عدة خصائص، وتكفينا الخاصية FILE_ATTRIBUTE_NORMAL
HANDLE hTemplateFile	0
) ;	

GetFileSize (	دالة الحصول على حجم الملف ودفعها إلى المسجل EAX
HANDLE hFile	مقبض الملف ويتم الحصول عليه من الدالة CreateFile
LPDWORD lpFileSizeHigh	0
) ;	

CreateFileMapping (	دالة لتجهيز الملف من أجل التخطيط في الذاكرة ودفع القيمة إلى المسجل EAX
HANDLE hFile	مقبض الملف ويتم الحصول من الدالة CreateFile

LPSECURITY_ATTRIBUTES lpFileMappingAttributes	0
DWORD flProtect	نوعية التعامل مع الملف، لنجعلها PAGE_READWRITE توجد خاصيتين أخريتين منها القراءة فقط وأيضا إمكانية النسخ
DWORD dwMaximumSizeHigh	0
DWORD dwMaximumSizeLow	حجم الملف
LPCTSTR lpName	0
);	

MapViewOfFile (	دالة التخطيط في الذاكرة
HANDLE hFileMappingObject	مقبض التخطيط المستخلص من دالة السابقة
DWORD dwDesiredAccess,	نوعية التعامل مع الملف لنجعله للقراءة والكتابة
DWORD dwFileOffsetHigh	0
DWORD dwFileOffsetLow	0
DWORD dwNumberOfBytesToMap	0
);	

WriteFile (	دالة لكتابة البيانات في ملف
HANDLE hFile	مقبض الملف
LPCVOID lpBuffer	عنوان المتغير الذي يحتوي البيانات المراد كتاباها
DWORD nNumberOfBytesToWrite	عدد البايتات التي يتم كتابتها
LPDWORD lpNumberOfBytesWritten	متغير لاستقبال عدد البايتات التي تم كتابتها
LPOVERLAPPED lpOverlapped	0
);	

UnmapViewOfFile (	دالة الانتهاء من التخطيط
LPCVOID lpBaseAddress	عنوان الملف المخطط في الذاكرة والذي تم الحصول عليه بالدالة MapViewOfFile
);	

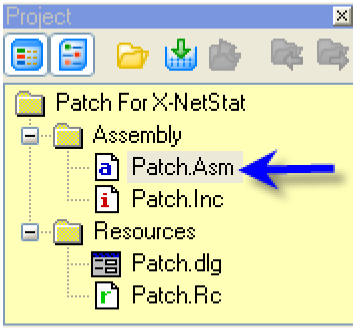
CloseHandle (	دالة إغلاق الملف
HANDLE hObject	مقبض الملف الذي تم الحصول عليه بالدالة CreateFile
);	

IsDlgButtonChecked (	التأكد من أن BackUp Control معلم وتدفع للمسجل EAX أحد القيمتين 0 أو 1

مقبض النافذة	HWND hDlg
ID الخاص بالـ BackUp Control	int nIDButton
	);

2.9 الشروع في برمجة الـPatch

الآن اختر قسم الكود من المشروع لكي نبدأ في كتابة Patch :



وهذا مقطع البيانات سواء بقيم مبدئية أو بدونها، وقد تم فيها تعريف بعض الرسائل ومسار الملف ومسار النسخة وبعض المتغيرات التي سنحتفظ بها لمعلومات الملف :

```
.data
FileName db "C:\Program Files\X-NetStat 5.0\xns5.exe",0
BackUpFile db "C:\Program Files\X-NetStat 5.0\xns5.exe.bak",0
BackUpText db "BackUp Created...",0
FileFound db "The file is Found",0
FileNotFound db "The File is Not Found",0
FilePatched db "The File Patched",0
.data?
hFile dword ?
sizeF dword ?
maping dword ?
nBytes dword ?
```

هذا المقطع الموافق للزر Patch وسيتم شرحه سطراً سطراً كي تفهم الفكرة وترسخ في ذهنك :

```
.elseif eax==WM_COMMAND
    .if wParam == 10
        invoke CreateFile,addr FileName,GENERIC_READ+ GENERIC_WRITE,FILE_SHARE_WRITE,0,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,0
        mov [hFile],eax
        cmp eax,-1
        jz Exit
        invoke GetFileSize,[hFile],0
        mov [sizeF],eax
        invoke CreateFileMapping,[hFile],0,PAGE_READWRITE,0,[sizeF],0
        invoke MapViewOfFile,eax,FILE_MAP_WRITE+ FILE_MAP_READ,0,0,0
        mov [maping],eax
        invoke IsDlgButtonChecked,hWin,13
        test eax,eax
        jz backupnC
        invoke CreateFile,addr BackUpFile,GENERIC_READ+ GENERIC_WRITE,FILE_SHARE_WRITE,0,CREATE_NEW,FILE_ATTRIBUTE_NORMAL,0
        mov edi,eax
        invoke WriteFile,eax,[maping],[sizeF],addr nBytes,0
        invoke CloseHandle,edi
        invoke SendDlgItemMessage,hWin,14,LB_ADDSTRING,0,addr BackUpText
        backupnC:
        mov eax,[maping]
        add eax,0A16C7h
        mov byte ptr [eax],74h
        invoke SendDlgItemMessage,hWin,14,LB_ADDSTRING,0,addr FilePatched
        invoke UnmapViewOfFile,[maping]
        invoke CloseHandle,[hFile]
        ret
        Exit:
        invoke SendDlgItemMessage,hWin,14,LB_ADDSTRING,0,addr FileNotFound
    .endif
```

invoke CreateFile, addr FileName, GENERIC_READ + GENERIC_WRITE, FILE_SHARE_WRITE, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0	دالة إنشاء أو قراءة ملف, وفي هذه الحالة قد جهزنا هذه الدالة بخاصية OPEN_EXISTING والسماح لنا بالقراءة والكتابة في الملف ودفع المقبض للمسجل EAX
mov [hFile], eax	بسبب تغير المسجلات سنقوم بحفظ مقبض الملف في المتغير hFile انظر قسم البيانات بدون قيمة مبدئية .data?
cmp eax, -1	المقارنة إن تم قراءة الملف أم لا؟ وذلك لأن هذه الدالة CreateFile تدفع للمسجل EAX القيمة – 1 إن لم يتم تنفيذه، أما بالنسبة للعكس فتقوم بدفع مقبض الملف للمسجل EAX
jz **Exit**	إن لم تنفذ هذه الدالة فسنخرج بسلام أما إذا نفذت فتابع الأسطر التالية
invoke GetFileSize, [hFile], 0	دالة جلب حجم الملف انطلاقاً من مقبضه الذي تم تخزينه في المتغير hFile ودفع حجم الملف للمسجل EAX
mov [sizeF], eax	الاحتفاظ بحجم الملف في المتغير sizeF انظر قسم البيانات بدون قيمة مبدئية .data?
invoke CreateFileMapping, [hFile], 0, PAGE_READWRITE, 0, [sizeF], 0	هذه الدالة من أجل أن تهيئ الملف للتخطيط في الذاكرة وهي بخاصية القراءة والكتابة، وتدفع مقبض التخطيط للمسجل EAX
invoke MapViewOfFile, eax, FILE_MAP_WRITE+ FILE_MAP_READ, 0, 0, 0	دالة التخطيط في الذاكرة، وستدفع للمسجل EAX عنوان الملف المخطط في الذاكرة وإن قمت بتنقيح الباتش وعمل

	بعد تجاوز هذه الدالة –للمسجل Follow in Dump – EAX ستجد في قسم Dump في المنقح شكل الملف وكأنك فتحته بـــ WinHex أو HexWorkshop
mov [maping], eax	الاحتفاظ بعنوان الملف المخطط في المتغير maping انظر قسم البيانات بدون قيمة مبدئية .data?
invoke IsDlgButtonChecked, hWin, 13	دالة للتأكد أن BackUp Control معلمة
test eax, eax	التأكد من أن EAX فيه إحدى القيمتين: صحيح أم خطأ
jz **backupnC**	اقفز إذا لم يكن BackUp Control معلم إلى العنوان الذي تم الرمز له بـــ backupnC
invoke CreateFile, addr BackUpFile, GENERIC_READ+ GENERIC_WRITE, FILE_SHARE_WRITE, 0, CREATE_NEW, FILE_ATTRIBUTE_NORMAL, 0	في حالة تم تعليم BackUp Control سنقوم بالاحتفاظ بنسخة من الملف وذلك بإنشاء ملف فارغ بهذه الدالة، وكما ترى فهي بخاصية CREATE_NEW
mov edi, eax	نقل مقبض الملف الجديد إلى المسجل EDI
invoke WriteFile, eax, [maping], [sizeF], addr nBytes, 0	كتابة بيانات الملف المراد كسره قبل التغيير في الملف الجديد، ويمكن أن تفعل مثل هذا الأمر بدالة CopyFile
invoke CloseHandle, edi	إغلاق الملف الجديد
invoke SendDlgItemMessage, hWin, 14, LB_ADDSTRING, 0, addr BackUpText	الكتابة في قائمة البيانات المعرفة في المتغير BackUpText في قسم .data
backupnC :	بدون تعليق
mov eax, [maping]	نعيد إلى المسجل EAX عنوان الملف المخطط في الذاكرة سابقاً
add eax, 0A16C7h	كما تعلم فإن هذا العنوان هو بداية الملف أي offset=0 ونحن نريد العنوان الذي يوافقه Offset =0A16C7h سنقوم بإضافة هذا الأخير إلى عنوان الملف المخطط
mov byte ptr [eax], 74h	سنقوم بنقل البايت كما غيرناها سابقاً ببرنامج OllyDbg من القيمة 75 إلى 47 في العنوان الموجود في المسجل EAX يمكنك إضافة ما تشاء من التعديلات بنفس الطريقة
invoke SendDlgItemMessage, hWin, 14, LB_ADDSTRING, 0, addr FilePatched	الكتابة في قائمة النص الموجود في المتغير FilePatched انظر قسم البيانات .data
invoke UnmapViewOfFile, [maping]	إنهاء الملف من الذاكرة
invoke CloseHandle, [hFile]	ثم إغلاق الملف
ret	الخروج بسلام
Exit :	في حالة استعصى فتح الملف أو عدم، فسيتم القفز إلى هذا العنوان من الأمر السابق jz Exit
invoke SendDlgItemMessage, hWin, 14, LB_ADDSTRING, 0, addr FileNotFound	الكتابة في قائمة النص الموجود في المتغير FileNotFound انظر قسم البيانات .data

وبهذا نكون قد أنهينا مقطع الكود الخاص بزر Patch.

لنمر إلى الجزء الخاص بمقطع كود كل من زري About و Exit. لن نشرحهما ولكن سنكتفي بوضع صورة لمقطع الكود؛ لأن الأمر سهل كما أنه يتوجب عليك معرفته من الجزء الأول من الكتاب.

الجزء الخاص بقسم البيانات في شكله الأخير :

```
.data
FileName db "C:\Program Files\X-NetStat 5.0\xns5.exe",0
BackUpFile db "C:\Program Files\X-NetStat 5.0\xns5.exe.bak",0
BackUpText db "BackUp Created...",0
FileFound db "The file is Found",0
FileNotFound db "The File is Not Found",0
FilePatched db "The File Patched",0

;=======Message Text==========
MessageAoubt db "Coded By Mouradpr/AT4RE",0
AboutTitre db "Message",0
;=======Message Text==========

.data?
hFile dword ?
sizeF dword ?
maping dword ?
nBytes dword ?
```

وهذا مقطع الكود الخاص بالزرين About و Exit :

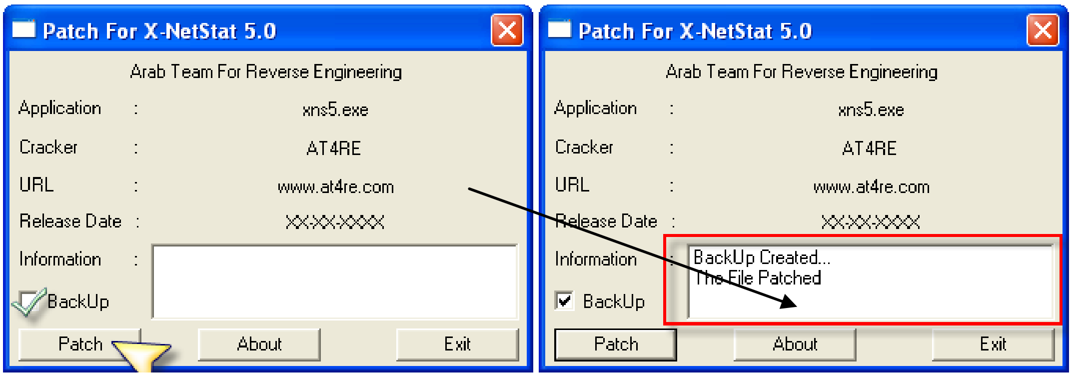
```
    .elseif wParam==11        ; <-- About
        invoke MessageBox,hWin,addr MessageAoubt,addr AboutTitre,MB_OK
    .elseif wParam==12        ; <-- Exit
        invoke ExitProcess,0
.endif
```

2.5. تجربة الـ Patch

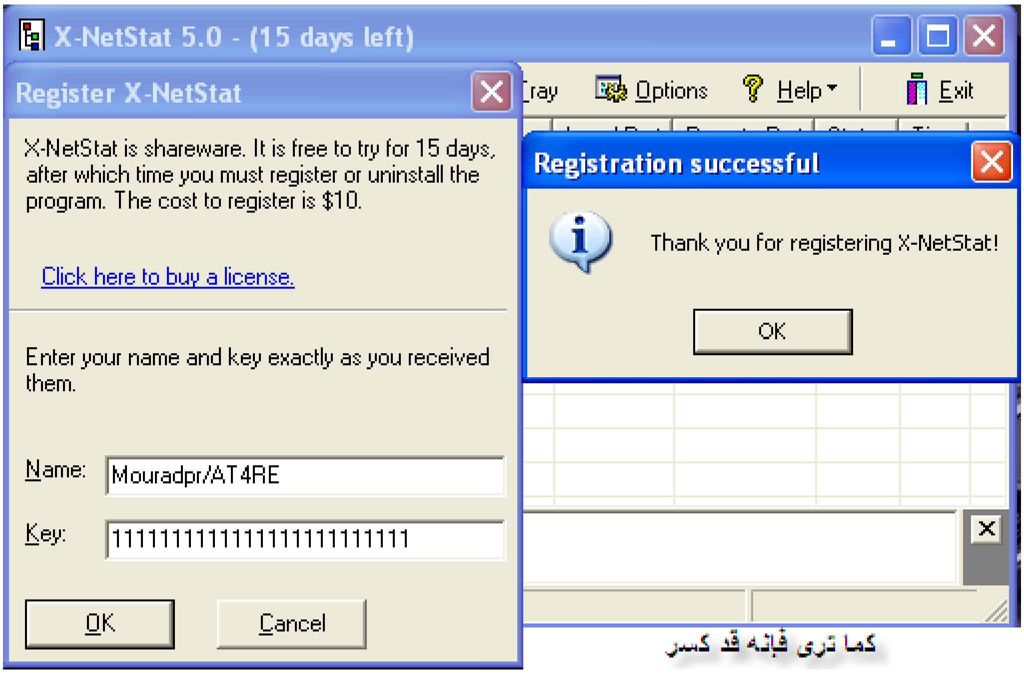
بعد إتمام برمجة الـــ Patch سنقوم بتجميعه و تجربته :



قم بالضغط على الزر Patch للتجربة، وأيضاً BackUp Control :



التأكد من تسجيل البرنامج بعد تطبيق الـــ Patch :

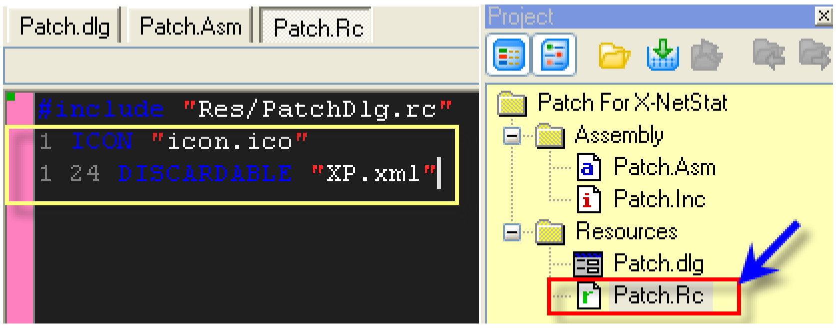


مبروك عليك فقد تم كسر البرنامج بالـــPatch الذي برمجناه.

2.6. إضافة تحسينات للـ Patch

2.6.1. إضافة XP Style والأيقونة

نقوم بنسخ الأيقونة وملف XP.xml المرفق مع الدرس إلى مجلد السورس كود ثم إضافته في المشروع كالتالي :



2.6.2. جعل ال PATCH شفاف ويخرج بشكل غير عادي

سنضيف إلى مقطع كود الــ Dialog اللؤد التالي :

```
.if eax==WM_INITDIALOG

    invoke SetFocus,eax
    invoke GetWindowLong,hWin,GWL_EXSTYLE
    or eax,WS_EX_LAYERED
    invoke SetWindowLong,hWin,GWL_EXSTYLE,eax
    invoke SetLayeredWindowAttributes,hWin,0,150,LWA_ALPHA
```

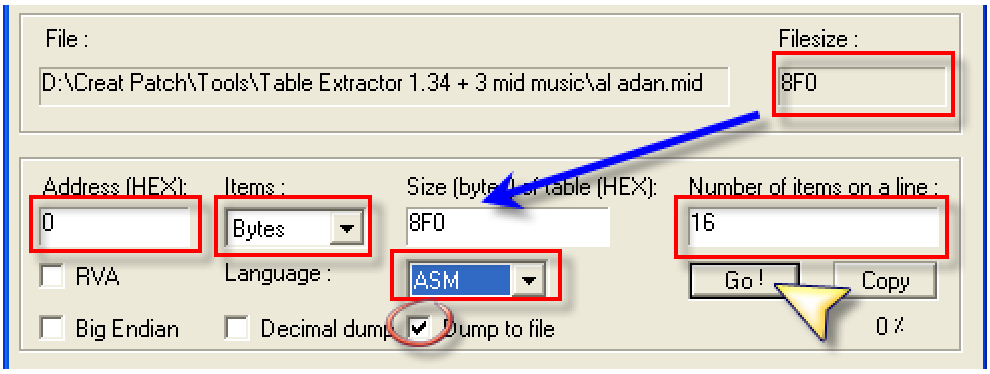
ثم إلى أمر الخروج كالتالي :

```
        .elseif wParam==12
            invoke AnimateWindow, hWin, 500,AW_HOR_POSITIVE or AW_HIDE
            invoke ExitProcess,0
    .endif

.elseif eax==WM_CLOSE
    invoke AnimateWindow, hWin, 500,AW_VER_POSITIVE or AW_HIDE
    invoke EndDialog,hWin,0
```

2.6.3. إضافة نشيد من نوع MID إلى ال PATCH

سنحتاج برنامج Table Extractor 1.34 لتحويل الملف إلى بيانات لدمجه مع الــ patch كالتالي :



بعد الضغط على زر Go! ستحصل على الملف بجانب البرنامج باسم result.txt ، قم بتغيير اسمه من أجل التنظيم فقط إلى Music.inc

ثم أضف الأكواد التالية :

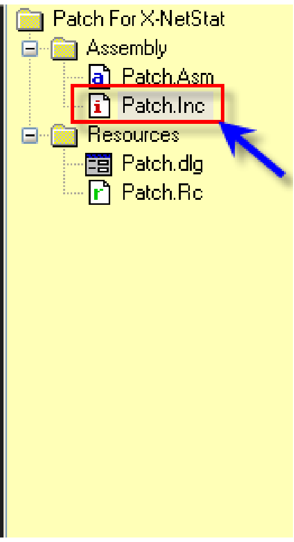
```
include windows.inc
include kernel32.inc
include user32.inc
include Comctl32.inc
include shell32.inc

include winmm.inc

includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib

includelib winmm.lib
DlgProc          PROTO    :HWND,:UINT,:WPARAM,:LPARAM

PlayMidiFile PROTO :DWORD,:DWORD
```



أما في القسم .code فقمت بإضافة بضعة أسطر لدالة قراءة الملفات من نوع Mid :

```
DlgProc endp
PlayMidiFile proc hWin:DWORD,NameOfFile:DWORD
    LOCAL    mciOpenParms:MCI_OPEN_PARMS,mciPlayParms:MCI_PLAY_PARMS

    mov      eax,hWin
    mov      mciPlayParms.dwCallback,eax
    mov      eax,OFFSET szMIDISeqr
    mov      mciOpenParms.lpstrDeviceType,eax
    mov      eax,NameOfFile
    mov      mciOpenParms.lpstrElementName,eax
    invoke mciSendCommand,0,MCI_OPEN,MCI_OPEN_TYPE or MCI_OPEN_ELEMENT,ADDR mciOpenParms
    mov      eax,mciOpenParms.wDeviceID
    mov      MidDeviceID,eax
    invoke mciSendCommand,MidDeviceID,MCI_PLAY,MCI_NOTIFY,ADDR mciPlayParms
    ret

PlayMidiFile endp
```

في قسم البيانات data. وأيضا ?data. ستضيف البيانات التالية :

```
szMIDISeqr           db 'Sequencer',0
include Music.inc
MusicFile db "damini.mid",0
.data?
MidDeviceID          dd ?
```

ثم في مقطع الكود الخاص بـ Dialog :

```
.if eax==WM_INITDIALOG
    invoke CreateFile,addr MusicFile,GENERIC_READ+ GENERIC_WRITE,FILE_SHARE_WRITE,0,CREATE_ALWAYS,FILE_ATTRIBUTE_HIDDEN,0
    mov edi,eax
    invoke WriteFile,eax,addr table,8F0h,addr nBytes,0
    invoke CloseHandle,edi
    invoke PlayMidiFile,hWin,addr MusicFile
```

أما القيمة 8F0h فهي حجم ملف النشيد الذي أضفناه, والذي كنا قد حصلنا عليه ببرنامج Table Extractor 1.34

وأخيراً قم بتجميع الـــ Patch :



جميل جداً أليس كذلك؟!

أرفقنا مثالين مع الدرس لدراستهم وإبداع شكل جديد وخصائص جديدة.

طريقة برمجة CrackME KeyGenMe بالأسمبلي واستغلال الخوارزميات

CrackMe, KeygenMe and CryptoMe Coding

موضوع هذه الفقرة يعتبر من أساسيات الهندسة العكسية؛ فالهدف من هذا العلم ليس كسر البرامج فقط، فهنالك الكثير مما يجب عليك تعلمه والاطّلاع عليه، ومن ذلك أن تنمي خبرتك في مجال تحسين حماية البرامج.

سيتناول الدرس المراحل التالية :

1) موجز في الحماية وحدودها.
2) دراسة نظرية لـ CrackMe .
3) دراسة تطبيقية لـ CrackME .
4) دراسة نظرية لـ KeyGenME .
5) دراسة تطبيقية لـ KeyGenME بسيط
6) مفهوم KeyFileME مع التمثيل.
7) خوارزميات التشفير

- MASM + RadAsm or WinAsm.
- IDAFicator و OllyDbg .
- AT4RE FastScanner ثم Kanal Plugin و Hasher Plugin .
- LordPE.

قد يتبادر للأذهان السؤال التالي: لماذا الأسمبلي؟

لا نقول إنها اللغة الوحيدة التي يمكن برمجة CrackMe بها، لكنها تبقى الأفضل نظراً لسهولة تنقيح برامجها.

1. موجز في الحماية وحدودها

لا ندري من السباق في الوجود التنقيح من أجل الكسر أم التنقيح من أجل تصحيح الأخطاء والحماية ؟! لكن ما يجب أن تتأكد منه أن الاستغلال الأصح هو من أجل التصحيح والحماية.

إن الهندسة العكسية في تطور مستمر، وفي كل يوم نجد شيئاً جديداً وحماية معقّدة، ولماذا الحماية ؟ الجواب بسيط: هو التجارة والربح والبيع، ولذلك تجد جميع البرامج التجارية محمية، وتتنوع حسب خبرة المبرمج وحالته المادية, فهناك برامج حماية غير مجانية؛ مثل: ExeCryptor وArmadillo وTheMida إلى غير ذلك من برامج الحماية الشهيرة التي طالما صادفتنا من قبل.

هناك عدة أشكال للحماية : مثلاً إضافة نافذة مزعجة بتسجيل أو تشفير لقيم PE أو ضغط الكود وغير ذلك.

موضوع الدرس سيكون حول نافذة التسجيل ومنها تم استقاء فكرة KeyGenME ، كما تم استقاء فكرة الــ CrackME من النافذة المزعجة التي تظهر عند إقلاع بعض البرامج.

حدود الحماية :

ألم تتساءل من قبل: لماذا تتعدد برامج الحماية بطرق جديدة؟ قد تقول: تم كسر السابقة. لكن لماذا؟

ذلك ما نريد التنويه له وهو حدود هذه الحماية حتى يتم كسرها.

هناك عدة أسباب، منها البرنامج في الذاكرة فهذا قد شكل عائقاً كبيراً في الميدان، ومع ظهور دوال التعديل في الذاكرة تم تجاوز هذا إلى شيء أكثر تطوراً.

ألم تتساءل من قبل: لماذا لا يكون هنالك تسجيلاً واحداً للبرنامج، بحيث يصعب حتى فتح البرنامج من دونه؟ لكن ماذا لو انتشر التسجيل؟ أكيد سيحل خراب بالشركة.

يجب أن يجرب البرنامج قبل أن يتم اقتناؤه ولذلك تجد عدة برامج Trial .

2. دراسة نظرية لـ CrackMe

ظهر الــ CrackMe لتمثيل بعض الحماية التي لا تطلب تسجيلاً، وإنما هناك بعض الأشياء المزعجة؛ كالنوافذ مثلا أو تفعيل أزرار أو قوائم، لكن لماذا وقد تم كسر الكل؟ نقول لك في هذا الوقت: نعم! لكن فيما سبق كانت هذه تعتبر أفكاراً متطورة.

حسناً سنقوم بتطبيق بعض الأفكار النظرية لبرمجة CrackMe :

- الفكرة الأولى : هي النافذة المزعجة، وتتمثل في إضافة نافذة Window أو Dialog قبل تنفيذ البرنامج الأصلي وهي مزعجة فعلاً.

ملاحظة هامة : الرسائل المزعجة جزء من النوافذ المزعجة أيضاً.

- الفكرة الثانية : وهي عدم تفعيل بعض الأزرار والقوائم، وهي فكرة بدائية شيئاً ما لكن مع ذلك تجدها شائعة وتتمثل في عدم تفعيل بعض الأزرار أو القوائم لحجب خاصية جيدة في البرنامج؛ كالحفظ مثلاً.

ملاحظة هامة : ليس شرطاً أن تفعّل النوافذ والقوائم فقط، فالأمر قد يشمل ما هو أكثر من ذلك.

- الفكرة الثالثة –وهي تعزيز للموضوع فقط لأن هناك الكثير من الطرق– : تتمثل في إضافة توقيت للبرنامج حيث يعمل دقيقة أو أقل أو أكثر.
- الفكرة الرابعة : حماية البرنامج بسريال واحد لا يوجد للبرنامج غيره.
- الفكرة الخامسة : البايتات المسروقة.
- الفكرة السادسة : حماية الكود بأحد الخوارزميات (سنتعامل مع CRC).

3. دراسة تطبيقة لـ CrackMe

تلزمك بعض الخبرات السابقة في دوال النظام؛ لذلك ننصحك بمراجعة بعض المواضيع المتخصصة أو المواقع مثل MSDN .

3.1. الفكرة الأولى : النافذة المزعجة

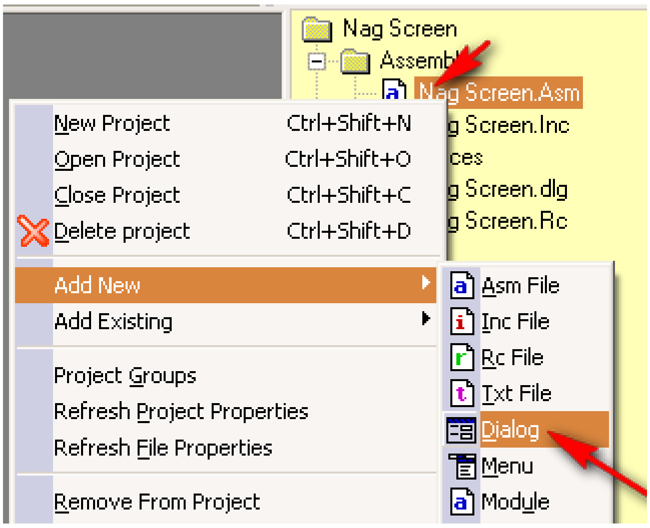
سنقوم بإضافة Dialog بأحد المحررات كـ RadAsm مثلاً، والذي سنستعمله طوال الدرس، لذلك ننصحك بتحميل بعض الدروس المتعلقة بذلك، والحمد لله فهي الآن موجودة في منتدى الفريق في دروس الـ Keygening وقد وضعنا البرنامج كاملاً مع MASM وبجميع المكتبات، تجد روابطه في بداية الدرس.

دعنا نبدأ : يفترض بك أنك قد جهزت Template فارغ بـ Dialog .

File → New Project → Win32 App (Project Name : Nag Screen) → DialogApp.tpl

حسناً الآن لدينا Dialog سنقوم بإضافة آخر باسم Nag وذلك كالتالي :

اضغط الزر الأيمن :



ثم احفظه باسم Nag .

هنالك شيء هام يجب أن تتذكره، وهو الـــ ID الخاص بكل Control أضفناه؛ مثلاً في هذه الحالة تجد أن الـــ ID للـــ Dialog الجديد هو : 1000 .

الآن سنجهز الـــ Nag بما نريد ولتكن هكذا :



شيء مضحك ☺ .

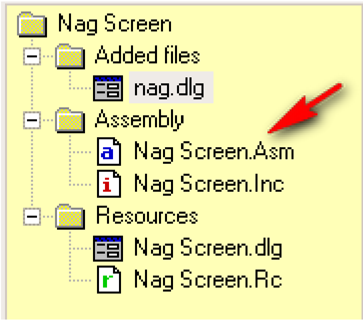
تحذير : تذكر أننا أضفنا هذا في النافذة الجديدة التي أضفناها وليس في الرئيسية.	

سنغير ID لزر Trial

Botton Trial ID = 90

سنقوم الآن بالانتقال إلى الدالة التي ستطبع لنا النافذة قبل دخول البرنامج.

انتقل إلى النافذة الرئيسية وضع ما تريد لبرنامجك –نحن سنتركها فارغة في مثالنا– ثم اختر ملف الكود :



تعريف دالة إظهار نافذة من الريسورس، وهي دالة مطورة سهلة الاستخدام:

DialogBoxParam(	دالة إظهار نافذة موجودة في الريسورس
HINSTANCE hInstance	مقبض الملف في الذاكرة
LPCTSTR lpTemplateName	الـ ID الخاص بـــ Dialog
HWND hWndParent	في حالتنا NULL لن نحتاجه
DLGPROC lpDialogFunc	الإجرائية التابعة للنافذة الجديدة
LPARAM dwInitParam	في حالتنا NULL لن نحتاجه
);	

حسناً سنضيفها في WM_INITDIALOG الخاص بالنافذة الرئيسية كالتالي :

```
DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
    mov     eax,uMsg
    .if eax==WM_INITDIALOG

        invoke DialogBoxParam,hInstance,1000,NULL,addr NagScreen,NULL
```

لا تسأل عن السهم الأول الذي يشير لـــ 1000 وإلا فيجب عليك إعادة الدرس فأنت لست متابعاً بشكل جيد!

أما NagScreen التي يشير لها السهم الأيمن فهي الإجرائية التي سنعطيها للنافذة المزعجة، وهي شبيهة بإجرائية النافذة الرئيسية، ونعطيها Prototype كالرئيسية :

```
include Nag Screen.inc
NagScreen           PROTO   :HWND,:UINT,:WPARAM,:LPARAM
.code

start:
```

أما أوامر الإجرائية فهي كالتالي :

```
DlgProc endp
NagScreen proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM
mov      eax,uMsg
.if eax==WM_COMMAND
    .if wParam==90
        invoke EndDialog,hWin,0
    .endif

.elseif eax==WM_CLOSE
        invoke EndDialog,hWin,0
.endif
xor eax,eax
ret
NagScreen endp
end start
```

بالنسبة للدالة EndDialog فهي لإنهاء النافذة انطلاقا من مقبضها.

الآن جمع الملف من هنا :  في الأعلى بعد Menu bare

بهذا نكون قد أنهينا فقرة النافذة المزعجة وستجد المثال مع السورس كود في pack الدرس هنا  تحت : [3.1]

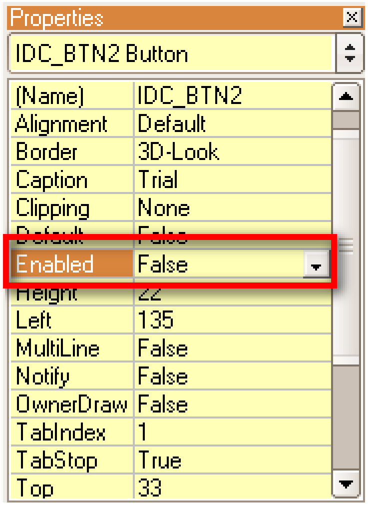
3.2 الفكرة الثانية : عدم تفعيل الأزرار

هذه الفكرة تعتمد على الريسورس فقط، وسنلجأ للأزرار فقط وسنترك لك القوائم.

قم بإنشاء مشروع جديد بنفس مواصفات المشروع السابق باسم Enable ، ثم أضف له بعض الأزرار أو أي شيءٍ تريده من الـــ Controls وليكن هكذا مثلا :



ثم اتجه للأدوات التي تريد إزالة خاصة التفعيل عنها وعدل خصائصها بهذا النحو :



جمع الملف وأنظر الشكل النهائي .

حتى الآن كل شيء على ما يرام وبسيط لكن كسر هذه الفكرة أمر بسيط جداً ويتم بأحد محررات الريسورس من ResHacker أو ما شابه.

الآن ماذا لو أردنا القيام بذلك برمجياً، وهو الأمر الأهم في هذه الفقرة. كل ما نحتاجه هو مقبض الزر أو أي Control آخر نريده، ثم نجلبه بواسطة الدالة GetDlgItem , تحتاج هذه الدالة لبارامترين: الأول هو مقبض النافذة التي تحتوي الـــ Control ، والثاني هو الـــ ID الخاص بالـــ Control . إن نجحت هذه الدالة فإنها ترجع مقبض الـــ Contorl في المسجل eax ، ثم لتفعيل أو عدم تفعيل الـــ Control نستدعي الدالة EnableWindow أيضاً ببارامترين: الأول هو مقبض الـــ Controls ، أما الثاني فهو أمر التفعيل أو عدم التفعيل FALSE or TRUS .

وبالتالي يكون الكود لدينا في هذا المثال لإلغاء تفعيل زر TRIAL ذي ID = 1002 كالتالي :

```
DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

    mov     eax,uMsg
    .if eax==WM_INITDIALOG
        invoke GetDlgItem,hWin,1002
        invoke EnableWindow,eax,FALSE
    .elseif eax==WM_COMMAND
```

قم بتجربة أي Control تريد، وبهذا الكود الصغير ننهي فقرة عدم تفعيل الأزرار.

الكود من هنا تحت [3.2]

3.3 الفكرة الثالثة : إضافة مؤقت يغلق البرنامج في زمن معين

قم بإنشاء مشروع جديد باسم Timer .

دالة وضع مؤقت في المشروع مثل ما نفعل في Delphi وVB نضيف Timer لكن الأمر يختلف هنا فنحن نضعه برمجياً	SetTimer (
مقبض النافذة التي سنضيف لها المؤقت	HWND hWnd
لن نحتاجه لذا أعطه القيمة 0	UINT nIDEvent
هذا هو المهم وهو مدة المؤقت وتحسب بأجزاء الألف للثانية	UINT uElapse
لا حاجة لنا به لذا أعطه القيمة 0	TIMERPROC lpTimerFunc
	);

سنضيف في هذا المثال مؤقتاً يغلق البرنامج بعد 5 ثوانٍ :

```
    mov     eax,uMsg
    .if eax==WM_INITDIALOG
        invoke SetTimer,hWin,0,5000,0

    .elseif eax==WM_TIMER
        invoke ExitProcess,0
```

WM_TIMER هي إجرائية المؤقت.

تجد سورس شامل للفكرة هنا تحت [3.2]

3.4 الفكرة الرابعة : حماية السريال

3.4.1 حماية بسريال قابل للعكس

هذه فكرة بسيطة تعتمد دالة المقارنة بين التسجيل المدخل والتسجيل الأصلي.

قم بإنشاء مشروع جديد باسم Mono Serial وجهز الـــ Template هكذا:



ملاحظة هامة : يجب أن تتذكر الـــ ID الخاص بكل Control .

في مثالنا هذا سنعطي :

ID Boutton Register = 1002
ID Edit Text = 1001

الفكرة العامة للمثال :

- جلب السريال المدخل من المستخدم.
- مقارنته بالأصلي بالدالة lstrcmp ، وهي ترجع قيمتين 0 حالة التطابق.
- إذا حصل التطابق خرجت رسالة الشكر Good Boy .
- إذا لم يحصل التطابق خرجت رسالة الخطأ Bad Boy .

كل مرحلة ترافقها دالة تم شرحها في النسخة السابقة من الكتاب؛ لذلك لا حاجة للإعادة وسنضع الكود مباشرة.

سنحتاج بعض المتغيرات كالمتغير الذي يحتوي السريال الصحيح والمتغير الذي سنحفظ فيه السريال المدخل ثم رسالة الشكر والخطأ.

```
.data
CorrectSerial db "Mouradpr/AT4RE",0
Good db "Good Work",0
TitreGood db "Success",0
bad db "Bad Boy",0
.data?
SerialUser db 32 dup(?)

.code
```

في إجرائية الأوامر سنضيف مقطع الكود لزر التسجيل :

```
.elseif eax==WM_COMMAND

    .if wParam==1002 ; Register Botton ID
        invoke GetDlgItemText,hWin,1001,addr SerialUser,32 ; Get Serial Form User
        invoke lstrcmp,addr SerialUser,addr CorrectSerial
            .if eax==0
                invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
            .else
                invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
            .endif
    .endif
```

انتهت الفقرة وستجد سورس شامل للفكرة هنا  تحت [3.4.1]

3.4.2 حماية بسريال غير قابل للعكس

هل صادفك مرة برنامج يأخذ السريال ولا يقارنه بنفسه؛ مثلاً يقارنه بشيء مرتبط به ؟!

في هذا المثال سنستعمل خوارزمية بسيطة جداً لتوضيح الأمر.

الخوارزمية ستعتمد مقارنة مجموع قيم Hex للحروف المدخلة مع قيمة معينة.

الصيغة الرياضية للخوارزمية:

```
I : integer
X : integer
For I = 0 to Len(Name)
X = X + Name(I)
```

لنطبق الفكرة على كلمة AT4RE : E = 45h ; R = 52h ; 4 = 34h ; T = 54h ; A = 41h

بجمع القيم نجد Algo (AT4RE) = 160h

الآن لنفترض أن السريال الصحيح هو AT4RE ما سيقوم به الــ CrackMe هو جمع قيم Hex الاسم المدخل ثم مقارنتها مع 160h.

هناك سر في الخوارزمية سنقوله لك بعد أن ننتهي من الكود البرمجي!!

قم بإنشاء مشروع جديد باسم Brute Serial بنفس خصائص الــ CrackME السابق بزر Register وEdit Text لجلب التسجيل المدخل. وبنفس الــ ID .

المتغيرات التي سنعتمد عليها هي نفس متغيرات مثالنا السابق تقريباً.

```
.data
Good db "Good Work",0
TitreGood db "Success",0
bad db "Bad Boy",0
.data?
SerialUser db 32 dup(?)
```

كود خوارزمية التحقق لزر التسجيل :

```
.elseif eax==WM_COMMAND
    .if wParam==1002
        invoke GetDlgItemText,hWin,1001,addr SerialUser,32 ; Get Serial Form User
        mov edi,eax
        xor esi,esi
        xor edx,edx
        SerialCheck:
        movsx eax,byte ptr [SerialUser+esi]
        add edx,eax
        inc esi
        cmp esi,edi
        jnz SerialCheck
        cmp edx,160h
        jz GoodMessage
        invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
        ret
        GoodMessage:
        invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
    .endif
```

لا يفترض بنا أن نشرحه لك إذا كنت قد قرأت النسخة الأولى من الكتاب وتابعت دورة الأسمبلي التي أقامها الفريق قي موقعه وإلا فنحن نبني في الفراغ.

جمع الملف وجرب. رائع!! أليس كذلك؟!

لكن هنالك سر في هذه الخوارزمية، حيث أنه توجد بها العديد من الأسماء التي مجموع حروفها يساوي 160h !!

جرب مثلاً السريال : mo**0

كل ما في الأمر هو تجربة العديد من الكلمات وإن تطابقت المقارنة فذلك أيضا سريال.

ملاحظة : الأمر بسيط لكسر هذه الخوارزمية ولا يحتاج لوقت طويل لكن هناك بعض الخوارزميات وللأسف تحتاج لأيام أو أشهر لفكها وستجد شيئاً من ذلك في طيات هذا الكتاب.	

وبهذه الملاحظة ننهي هذه الفقرة، وستجد سورس شامل للفكرة هنا تحت [3.4.2]

3.5 الفكرة الخامسة : البايتات المسروقة

هذه أفضل تقنية في الحماية ولكن للأسف لها مساوئ سنعرفها في نهاية الدرس.

هذه الفقرة تتطلب بعض الخبرات الأساسية في OP Code أو الترجمة الهكسية لأوامر الأسمبلي، إنها تسميتنا لها فقط.

مراحل الفلكوة :

- تحديد المكان المناسب للحماية.
- تحويل الـــ OP Code إلى سريال.
- تحديد حجم الخوارزمية وتخريب الكود.
- خوارزمية إدراج السريال كــ OP Code الأصلي وجعل قسم الكود قابل للكتابة والقراءة.

- التجربة.

3.5.1. تحديد المكان المناسب للحماية

سنستعمل المثال السابق كتطبيق لهذه الفقرة، لذلك ننصحك بأخذ نسخة منه لتطبيق هذه الفقرة باسم X-CrackMe ثم افتح المشروع لكي نبدأ.

تحديد المكان المناسب للحماية ليس بالأمر العشوائي والبسيط، بل يتطلب تنقيح البرامج لمعرفة المناطق الحساسة بها، لكننا في هذه الفقرة سنقدم مثالاً بسيطاً تفهم منه ذلك.

> ملاحظة هامة : إن التعبير: xxxxx في MASM يشير إلى عنوان.

سنقوم بحصر الخوارزمية بين عنوانين للحصول على حجم الـــ OP Code :

```
.elseif eax==WM_COMMAND
    .if wParam==1002
        invoke GetDlgItemText,hWin,1001,addr SerialUser,32 ; Get Serial Form User

        sizeOPCode equ (endStolenBytes - StolenBytes)

        StolenBytes:

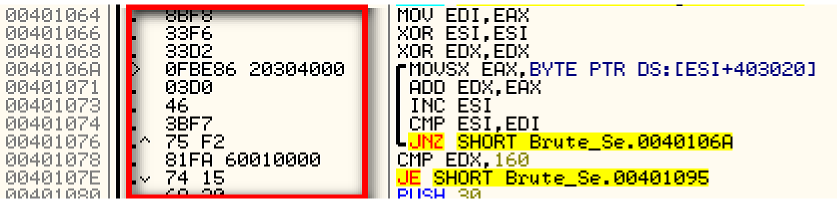
        mov edi,eax
        xor esi,esi
        xor edx,edx
        SerialCheck:
        movsx eax,byte ptr [SerialUser+esi]
        add edx,eax
        inc esi
        cmp esi,edi
        jnz SerialCheck
        cmp edx,160h
        jz GoodMessage

        endStolenBytes:

        invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
        ret
        GoodMessage:
        invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
```

انظر للطريقة التي حصرنا بها الخوارزمية والحصول على حجم OP Code , إذن لدينا في sizeOPCode حجم الـــ OP Code الآن وحصرنا الخوارزمية بين عنوانين رمزنا لهما بـــ StolenBytes و endStolenBytes .

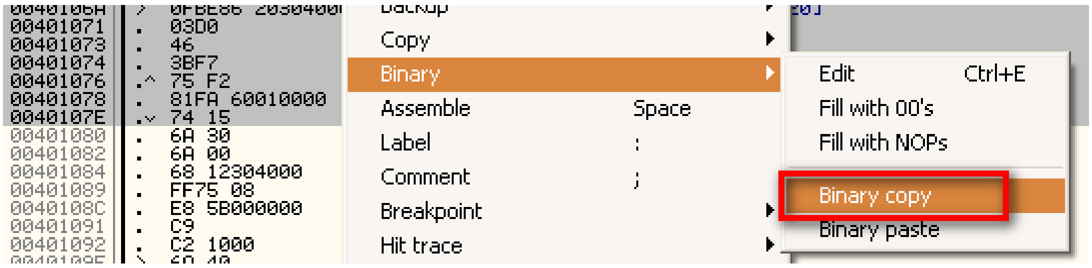
نظرة عن قرب لمعنى الــ OP Code في برنامج التنقيح OllyDbg :



تلك البايتات المحاطة باللون بالأحمر هي ما نعني. إذن السريال الذي يجب أن ندخله ما هو إلا أكواد برمجية لإكمال البرنامج وفي حالة كانت خطأ سيغلق البرنامج بالخطأ.

3.5.2. تحويل الـ OP CODE إلى سريال

لا ندري إن فاتك هذا، لكنه أمر مهم ولا يحتاج لشرح كثيف، كل ما نحتاجه هو أننا سنقوم بتظليل الخوارزمية في برنامج التنقيح ونسخ بايتات الكود هكذا :



نلصق البيانات في المفكرة لتذكره, لا بأس أن نضعها هنا :

8B F8 33 F6 33 D2 0F BE 86 20 30 40 00 03 D0 46 3B F7 75 F2 81 FA 60 01 00 00 74 15

جميل جداً أليس كذلك؟! حسناً الآن سنحذف الفراغات :

8BF833F633D20FBE862030400003D0463BF775F281FA600100007415

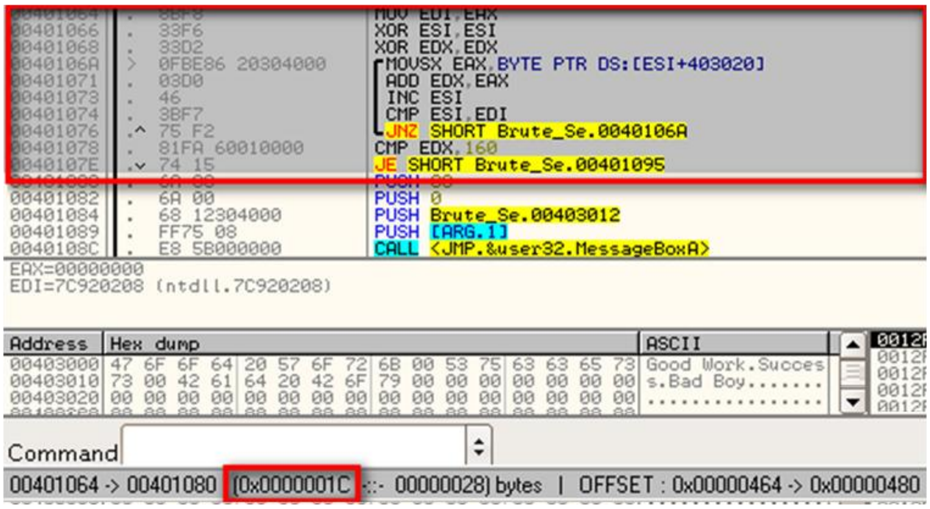
و الآن ها قد حصلنا على السريال للخوارزمية.

3.5.3. تحديد حجم الخوارزمية وتخريب الكود الأصلي

كل ما هو صعب سنبسطه لك، لذا لا تقلق أبداً !!

يوجد طريقتان لتخريب الكود: إما عن طريق OllyDbg وإما عن طريق برمجة أكواد مخالفة عن طريق RadAsm أو وضع مكان فارغ بحجم الخوارزمية. سنتطرق للأخيرة فقط لأننا نفضلها.

المشكلة هي في الحصول على حجم الـ OP Code وفي حالتنا سنستعمل إضافة من برمجة الأخ Zool@nder لتحديد حجم الكود... و الطريقة كالتالي : ما عليك إلا تظليل الخوارزمية ثم رؤية الحجم في الأسفل من خلال برنامج OllyDbg



حجم الخوارزمية كما تشاهد هو 1Ch ولو قمت بحساب طول السريال لوجدته 28d هكس.

الآن سنجعل المكان فارغ بإضافة متغير فارغ بحجم 1Ch :

```
sizeOPCode equ (endStolenBytes - StolenBytes)
StolenBytes:

db 1Ch dup(?)

endStolenBytes:
```

3.5.4. خوارزمية إدراج السريال ك OP Code

هذا هو الأمر المهم في هذه الفكرة؛ لأنه ستواجهنا مشكلة في تنفيذ الملف, ولكن لا عليك سندرج خوارزمية حقن السريال كـ OP Code لذلك سنحتاج إحدى الدوال لتحويل String to Hex وهي دالة htodw الموجودة في مكتبة الـ masm32 لذلك سنضمها في المشروع كالتالي:

```
include masm32.inc
includelib masm32.lib
```

ثم نضع خوارزمية حقن السريال المدخل كـ OP Code :

```
.elseif eax==WM_COMMAND
    .if wParam==1002

        invoke GetDlgItemText,hWin,1001,addr SerialUser,60 ; Get Serial Form User
        sizeOPCode equ (endStolenBytes - StolenBytes)

        xor esi,esi
        xor edi,edi
        InjectCode:
        invoke lstrcpyn,addr towBytow,addr [SerialUser+esi],3
        invoke htodw,addr towBytow
        mov byte ptr [StolenBytes+edi],al
        add esi,2
        inc edi
        cmp edi,sizeOPCode
        jnz InjectCode

        StolenBytes:

        db 1Ch dup(?)

        endStolenBytes:

        invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
        ret
        GoodMessage:
        invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
```

أما البيانات فكالتالي :

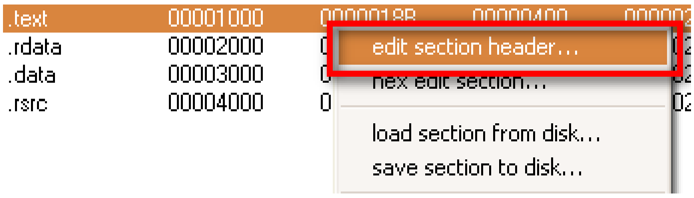
```
.data
Good db "Good Work",0
TitreGood db "Success",0
bad db "Bad Boy",0
.data?
SerialUser db 60 dup(?)
towBytow db 3 dup(?)
.code
```

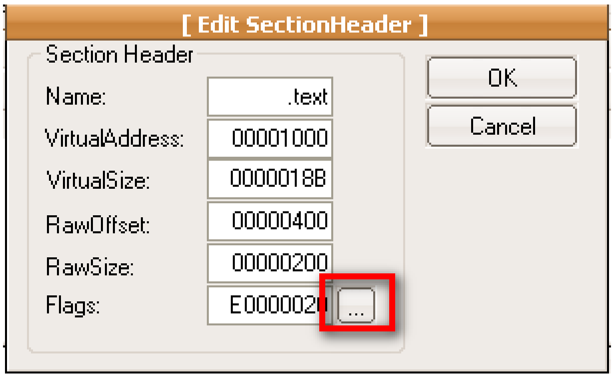
هذا هو الكود الأخير لحقن الكود، لكن ستواجهنا مشكلتين:

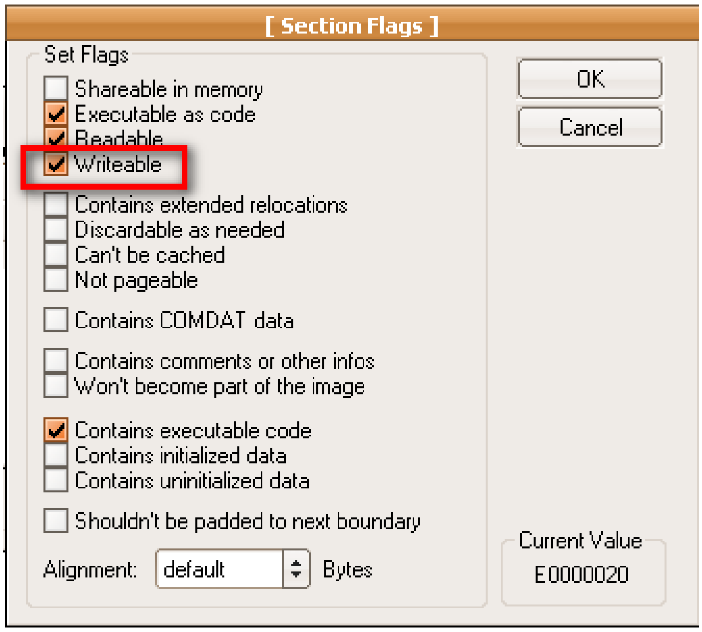
الأولى: أن الخانة لا تقبل ذلك الكود بأكمله لذلك سنزيد من طولها كحل سريع. والآن جمع الملف وضع السريال.. خطأ!! رغم أن وضع السريال الصحيح؟

الثانية: أنه لا يمكننا التعديل في قسم الكود!! لكن يمكن جعله للكتابة أيضاً بواسطة برنامج تعديل PE ونحن سنستعمل LordPE.

افتح الـ CrackME بـ LordPe ثم اتجه لـ Section ثم اضغط بالزر الأيمن على قسم الكود واختر :







3.5.5 التجربة

ما يميز مثل هذه الخوارزمية أنها فعالة بحيث لا يمكن تجاوزها إن استغلت بشكل مثالي، كتخريب كود خاصية الحفظ في برنامج ما.

ضع سريال خاطئ سيغلق بالخطأ، أما السريال الصحيح فسيمر بسلام.

لكن هنالك بعض المساوئ لهذه الحماية وهو أنه لا يمكن تجربة البرنامج إلا بالسريال الصحيح، وإذا انتشر السريال فلا حول ولا قوة إلا بالله.

هذه كانت فكرة برنامج الحماية الخاص بالفريق لقد بدأناها لكنها لم تكتمل، سنكمله في وقت لاحق إن شاء الله.

بهذه الملاحظة تنتهي هذه الفقرة، وستجد سورس شامل للفكرة هنا تحت [3.5.5]

3.6 الفكرة السادسة : حماية الكود بأحد الخوارزميات [سنتعامل مع CRC32]

لقد تطرقنا لهذا الأمر في النسخة الأولى من الكتاب ولا بأس أن نعيدها في هذا الدرس نظراً لأهميتها.

لماذا CRC ؟ سنقول لك: لا ندري لماذا!! لكن ما نتأكد منه أنه يمكنك أن تحمي برنامجك بنفس خاصية CRC32 Check وبخوارزمية من برمجتك.

هل تتذكر مثال السريال الغير قابل للعكس؟ تماماً مثل ذلك سنفعل، لكن هذه المرة لن نطلب سريال ولا هم يحزنون بل سنضيف مؤقتاً يتحقق من أن قيمة CRC ثابتة في كل جزء من الألف للثانية.

هذه الطريقة جيدة أيضا إن استغلت بشكل صحيح.

أتذكر المثال Mono Serial ؟ جميل قم بنسخه وسمه CRC32 Check فسنعمل عليه.

ملخص للفكرة :

- سنقوم بحساب قيمة CRC32 لما نريده في بداية البرنامج لكي تبقى ثابتة.
- ثم نضيف مؤقتاً نضع في إجرائيته حساب CRC32 كل جزء من الألف للثانية ومقارنته بالأصلي.

لحساب CRC هناك عدة مكتبات تتيح ذلك، وهناك أيضا الكود المصدري لها مفتوح، وستجد كل شيء مرفق مع الدرس. سنستعمل مكتبة ظهرت حديثاً فيها أغلب خوارزميات التشفير والهاش، وهي CryptoHash في الأسمبلي, بها العديد من الدوال وسنعطيك الـــ Prototype لخوارزمية CRC32 :

```
CRC32 proto lpBuffer:DWORD,dwBufLen:DWORD,dwCRC:DWORD; init dwCRC = 0
```

كل ما تحتاجه هذه الدالة هو المتغير الذي سنحدد CRC له ثم حجمه، أما الحفظ فهي تحفظه في المسجل eax .

لنضمن المكتبة كالتالي :

```
include Mono Serial.inc

include cryptohash.inc
includelib cryptohash.lib

.data
```

3.6.1. حساب CRC لإجرائية زر التسجيل

أولا يجب أن نحصر المكان الذي نريد عمل CRC Check له كما فعلنا في السابق بين عنوانين محددين كالتالي :

```
.if wParam==1002 ; Register Botton ID

    BeginCRCCheck:

    invoke GetDlgItemText,hWin,1001,addr SerialUser,32 ; Get Serial Form User
    invoke lstrcmp,addr SerialUser,addr CorrectSerial
        .if eax==0
            invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
        .else
            invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
        .endif

    EndCRCCheck:

.endif
```

ثم نضيف المؤقت ودالة حساب CRC في uMsg == WM_INITDIALOG كالتالي :

```
mov     eax,uMsg
.if eax==WM_INITDIALOG
    CRCSize equ (EndCRCCheck - BeginCRCCheck)
    mov ebx,CRCSize
    mov eax,offset BeginCRCCheck
    invoke CRC32,eax,ebx,INIT_CRC32
    mov OriginalCRC,eax
    invoke SetTimer,hWin,0,1,0
```

ثم في إجرائية المؤقت نضيف تأكد مستمر وأمر الخروج إذا تغير الـــ CRC ، لكن قبل ذلك لا بأس أن نوضح بعضاً من أجزاء هذا الكود الأخير:

CRCSize equ (EndCRCCheck – BeginCRCCheck)	;	عرفنا متغير به حجم المكان الذي نريد حمايته من التعديل
mov ebx,CRCSize	;	نقلنا الحجم للمسجل ebx لأن الدالة CRC32 لن تقبله إذا وضعنا

CRCSize بها

نقل عنوان بداية المكان الذي نريد حمايته للمسجل eax لنفس السبب السابق	;	mov eax,offset BeginCRCCheck
استدعاء الدالة لحساب CRC للمكان المراد حمايته أما INIT_CRC32 فهي ثابتة وتساوي صفر	;	invoke CRC32,eax,ebx,INIT_CRC32
ثم نحفظ قيمة CRC للمكان المراد حمايته في متغير باسم OriginalCRC وبحجم dword تجده في قسم البيانات	; ;	mov OriginalCRC,eax .data? OriginalCRC dd ?
وأخيراً دالة وضع المؤقت في المشروع	;	invoke SetTimer,hWin,0,1,0

أما ما يخص إجرائية المؤقت فهي فقط حساب متكرر لـ CRC لنفس المكان ومقارنته مع المكان الأصلي كالتالي :

```
.elseif eax==WM_TIMER
    CRCSize equ (EndCRCCheck - BeginCRCCheck)
    mov ebx,CRCSize
    mov eax,offset BeginCRCCheck
    invoke CRC32,eax,ebx,INIT_CRC32
    cmp eax,OriginalCRC
    jnz Exit
    ret
    Exit:
    invoke ExitProcess,0
```

ملاحظة هامة : كما نعلم فإن الـ Break Point (BP) في المنقحات ما هي إلا استدعاء للمقاطعة int3 ، وبالتالي سيكون نتيجة وضع نقطة توقف أن الكود سيتغير في المكان الذي حميناه وبالتالي سيغلق البرنامج عند وضع BP.

تجد سورس شامل للفكرة هنا تحت [3.6.1]

وبهذه الملاحظة ننهي فقرة الـ CrackME نظرياً وتطبيقياً بأمثلة بسيطة قابلة للتطوير، وستجد بإذن الله بعض الأمثلة في أحسن الاستغلال لدراستها.

4. دراسة نظرية لـ KeyGenMe

كما ذكرنا من قبل فإن فكرة الـ KeyGenME تم استقاءها من نافذة التسجيل لبرامج تجارية، لذلك فلن نعقّد الأمر وسنذكر بعض الاختلافات بين الـ CrackME والـ KeyGenME

CrackME	KeyGenME
التعامل مع الريسورس أزرار – قوائم – صور مزعجة ، و توقيت.	لا علاقة له بالريسورس
اعتماد بعض الخوارزميات للتضليل كـ CRC	لا علاقة له بالأمر إلا لمن يريد استعراض عضلاته
نسبة المنطق الرياضي به تكاد تنعدم ويعتمد غالباً على التجربة	منطق رياضي يقوم على علاقات رياضية محضة
ممل قليلاً ويتطلب خبرة وتجربة	ممتع ويتطلب خبرة رياضية بالإضافة إلى التجربة في الميدان

ما يميز الـ KeyGenMe في الغالب دالة ترابط بين عدة متغيرات ومجاهيل لأجل هدف واحد وهو توليد السريال, الصيغة الرياضية لدالة الـ KeyGenMe بصفة عامة :

```
Serial = f(x,y,z,….)
```

قد تدخل عدة عوامل مثل : اسم المستخدم , اسم الجهاز, الـ ID … إلى غير ذلك من العوامل وهي لا تعد ولا تحصى.

5. دراسة تطبيقية لـ KeyGenMe بسيط

نبدأ بالبسيط لبلوغ الهدف إن شاء الله. ما سنقوم به الآن هو برمجة KeyGenME بسيط إلى درجة تحت الصفر بإمكان أي شخص مبتدئ برمجة KeyGen له.

5.1. المثال الأول : المستوى تحت الصفر

فكرة المثال الأول :

- خانتين واحدة للاسم والأخرى للتسجيل وزر التسجيل.
- جلب الاسم.

- جعل الاسم Upper (حروف كبيرة) بالدالة CharUpper وهي مباشرة.
- جلب التسجيل.
- مقارنته مع الاسم بعد تكبير الحروف.
- النتيجة.

حسناً سنبدأ الآن بإنشاء مشروع جديد باسم First KeyGenMe أو أي اسم آخر تريده.

لنجهز Template بسيط كهذا مثلاً :



اجعل الـــ ID للـــ Controls كالتالي :

ID Serial Edit = 1001
ID Name Edit = 1002
ID Register Boutton = 1005

ادخل مباشرة لوضع كود زر التسجيل :

```
.elseif eax==WM_COMMAND
    .if wParam==1005 ; Register Boutton

        invoke GetDlgItemText,hWin,1001,addr NameBuffer,32
        invoke CharUpper,addr NameBuffer
        invoke GetDlgItemText,hWin,1002,addr SerialBuffer,32
        invoke lstrcmp,addr NameBuffer,addr SerialBuffer
            .if eax==0
                invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
            .else
                invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
            .endif
    .endif
```

أما المتغيرات فهي كالتالي :

```
.data
Good db "Good Work",0
TitreGood db "Success",0
bad db "Bad Boy",0
.data?
NameBuffer db 32 dup(?)
SerialBuffer db 32 dup(?)
.code
```

الأمر بسيط إلى الآن, ومبارك عليك أول KeyGenMe .

تجد سورس شامل للفكرة هنا تحت [5.1]

5.2. المثال الثاني : المستوى الأول

سنقوم ببرمجة KeyGenMe بمستوىً يزيد دراجات عن السابق لكنه يبقى بسيطاً.

فكرة المثال :

- جلب الاسم.
- تحويل كل حرف منه إلى قيمته بالـ Hex وإضافة قيمة معينة لها، وسنستعمل دالة تم شرحها في النسخة الأولى وهي wsprintf.
- جلب السريال.
- مقارنة السريال بناتج الخوارزمية.
- النتيجة.

قم بإنشاء مشروع جديد باسم KeyGenMe Level 1

ثم جهز الـ Template كالمثال السابق بنفس الـ ID للـ Controls ولنبدأ :

ندخل لكود زر التسجيل مباشرة :

```
.elseif eax==WM_COMMAND
    .if wParam==1005 ; Register Boutton

        invoke GetDlgItemText,hWin,1001,addr NameBuffer,32
        mov edi,eax
        xor esi,esi
        xor ebx,ebx
        NextChar:
        xor eax,eax
        mov al,byte ptr [NameBuffer+esi]
        add al,5Ch
        invoke wsprintf,addr [CorrectSerial+ebx],addr Hex,eax
        add ebx,2
        inc esi
        cmp esi,edi
        jnz NextChar
        invoke GetDlgItemText,hWin,1002,addr SerialBuffer,32
        invoke lstrcmp,addr CorrectSerial,addr SerialBuffer
            .if eax==0
                invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
            .else
                invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
            .endif
    .endif
```

ما سنضيفه هنا فقط أن القيمة التي أضفناها لكل حرف هي 5Ch وهي اختيارية.

أما المتغيرات فكالتالي :

```
.data
Good db "Good Work",0
TitreGood db "Success",0
bad db "Bad Boy",0
Hex db "%.2X",0
.data?
NameBuffer db 32 dup(?)
SerialBuffer db 32 dup(?)
CorrectSerial db 64 dup(?)
.code
```

تجد سورس شامل للفكرة هنا  تحت [5.2]

5.3. المثال الثالث : استعمال اسم الجهاز في توليد السريال

مازلنا في الأمثلة البسيطة حتى نمر على الأساسيات، وما قدمته إلى الآن ما هو إلا أفكار ابتدائية حتى تتيسّر لنا الفرصة لمعرفة الخدع والأفكار المتوسطة إلى المتقدمة في حماية برامجنا.

يلزمنا في هذه الفقرة دالة جلب اسم الجهاز وهي دالة مهمة للحصول على أفضل حماية.

جلب اسم الجهاز	GetComputerName (
متغير لحفظ اسم الجهاز بعد تنفيذ الدالة	LPTSTR lpBuffer
مساحة كبيرة نوعاً ما تحتاجها الدالة	LPDWORD nSize
	) ;

لن نستغلها الآن بشكل جيد؛ لأنه مازال يلزمنا بعض المعطيات التي سنتطرق لها في فقرات قادمة بإذن الله.

كل ما سنفعله سيكون على شكل المستوى التالي والبسيط :

- جلب الاسم من المستخدم.
- إضافة الاسم إلى اسم الجهاز بطريقة معينة.
- جلب التسجيل ثم مقارنته مع اسم الجهاز بعد إضافة اسم المستخدم له.

قم بعمل مشروع جديد وليكن باسم NameCumputer وجهز Template كالسابقة بها خانتين واحدة للاسم والأخرى للتسجيل، ثم زر التسجيل، واجعل ID للـ Controls كالتالي :

```
Name Edit (ID)          = 1001
Serial Edit (ID)        = 1002
Register Boutton (ID) = 1003
```

أما الكود فسنضعه كسابقه وسنشرح بعضه :

```
.elseif eax==WM_COMMAND
    .if wParam==1003 ; Register Boutton

        invoke RtlZeroMemory,addr ComputerName,sizeof ComputerName
        invoke GetComputerName,addr ComputerName,addr BufferSize
        invoke GetDlgItemText,hWin,1001,addr NameBuffer,32
        invoke lstrcat,addr ComputerName,addr addToName
        invoke lstrcat,addr ComputerName,addr NameBuffer
        invoke GetDlgItemText,hWin,1002,addr SerialBuffer,32
        invoke lstrcmp,addr ComputerName,addr SerialBuffer
            .if eax==0
                invoke MessageBox,hWin,addr Good,addr TitreGood,MB_OK+MB_ICONINFORMATION
            .else
                invoke MessageBox,hWin,addr bad,0,MB_OK+ MB_ICONEXCLAMATION
            .endif
    .endif
```

لا يوجد شيء جديد!! إذا استغربت من هذه الدالة RtlZeroMemory فلا داعي للغرابة، فما هي إلا دالة تقوم بتصفير متغير ما، ولها بارامترين: الأول هو المتغير، أما الثاني فهو حجم المتغير, لكن السؤال الذي يطرح نفسه : لماذا نستعملها ؟ سبب استعمالها متعلق باستعمال الدالة lstrcat التي تلصق نصي متغيرين، لكن عيبها أنها تضع الناتج في المتغير الأول، لذلك إذ استعملنا المثال عدة مرات سنحصل على فيض في المتغيرات، ولتفادي ذلك قمنا في كل مرة بتصفير المتغير الذي نلصق الاسم به.

الفكرة العامة للمثال : جلب اسم الجهاز ثم لصقه بـــ -AT4RE– ثم لصقه بالاسم المدخل من المستخدم ومقارنة الناتج بالتسجيل المدخل من المستخدم.

أما البيانات فكالتالي :

```
.data
Good db "Good Work",0
TitreGood db "Success",0
bad db "Bad Boy",0
addToName db "-AT4RE-",0
BufferSize dd  512

.data?
NameBuffer db 100 dup(?)
ComputerName db 100 dup(?)
SerialBuffer db 100 dup(?)
```

وبذلك ننهي المثال.

تجد سورس شامل للفكرة هنا  تحت [5.3]

6. مفهوم ومثال KeyFileME

هذه الفقرة يبدوا أنه لا أهمية لها في هذا الدرس، لكننا أضفناها ليكون الموضوع وشاملاً من جميع جوانبه.

KeyFileME : هو كسابقه مستقىً من بعض الحمايات التي لا تطلب اسماً ولا سريالاً أثناء تصفح البرنامج، لكنها تتحقق من ذلك في ملف أو في الريجستري.

تطرقنا لشرح الدوال المستعملة في هذه الفقرة في العديد من المرات ولن نعيدها، لذلك سندخل في المثال مباشرة.

سنقوم ببرمجة بريمج صغير يعمل لمدة 10 دقائق ثم يخرج إذا لم يجد الملف المسجل بجانبه.

خطوات المثال :

- إضافة مؤقت للبرنامج يعمل بعد 10 دقائق.
- إضافة خوارزمية البحث عن الملف والتحقق من أنه مسجل.

كل ما عليك فعله هو أن تضع مشروعاُ جديداً باسم KeyFileMe به نافذة مثلاً و بها كل ما تريد ما تريد.

ثم الكود وهو بسيط جداً :

```
.if eax==WM_INITDIALOG
    invoke SetTimer,hWin,0,10000,0

.elseif eax==WM_TIMER
    invoke CreateFile,addr RegFile,GENERIC_READ,FILE_SHARE_READ,0,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,0
    .if eax!=-1
        mov hFile,eax
        invoke ReadFile,hFile,addr RegBuffer,sizeof CorrectSerial,addr nBytes,0
        invoke lstrcmp,addr RegBuffer,addr CorrectSerial
            .if eax!=0
                invoke ExitProcess,0
            .endif
        invoke CloseHandle,hFile
    .else
        invoke ExitProcess,0
    .endif
```

أما البيانات :

```
.data
RegFile db "AT4RE.bin",0
CorrectSerial db "Mouradpr/AT4RE",0

.data?
hFile dd ?
nBytes dd ?
RegBuffer db 20 dup(?)
```

المثال مرفق مع الدرس.

تجد سورس شامل للفكرة هنا تحت [6]

إلى الآن كل شيء بسيط ويعتمد دالة المقارنة بصفة واضحة وسهلة الكسر.

7. خوارزميات التشفير والهاش

هذه هي الفقرة المميزة في الدرس والتي سنقوم بتفصيل أمثلتها بشكل يسهل إدراكه إن شاء الله.

سنقسم الفقرة إلى وحدات يمكن تلخيصها فيما يلي :

- خوارزميات التشفير.
- مثال لخوارزمية التشفير.
- خوارزميات الهاش.
- أمثلة لخوارزمية الهاش.
- الواضح في استغلال RSA .

7.1 خوارزميات التشفير

هناك عدة تعاريف لخوارزميات التشفير أو CIPHERS لكن سنعطي وجهة نظرنا فيها بطريقة مبسطة :

خوارزميات التشفير هي اسم مركب من : "خوارزميات" و"تشفير"

خوارزميات : جمع خوارزمية وهي كلمة مشتقة من اسم العالم الرياضي العربي المسلم الخوارزمي، وقد أطلق اسم خوارزمية على الطريقة العقلانية في تنظيم وحل المشاكل.

أما تشفير: فهي مشتقة من كلمة شفرة وتعني رمز، ويمكن أن نسميها أيضاً ترميز، أما الهدف من التشفير فهو قلب أو تخريب الرموز بشكل منتظم ويمكن فكه.

7.2. مثال لخوارزمية التشفير

هناك عدة خوارزميات لتشفير البيانات مثل Caesar , AES أو Way-3 أو Cast , DES , Mars .... إلى غيرها من الخوارزميات المشهورة، لكن لن نتطرق لشرحها لأنها ضخمة جداً ومعقدة شيئاً ما، لذلك سنكتفي بشرح أبسطها.

7.2.1. Caesar Cipher

هذه الخوارزمية قديمة وبسيطة من اختراع Caesar لذلك سميت باسمه. وهي تعتمد على إزاحة الحروف بمفتاح معين.

الصيغة الرياضية للخوارزمية :

Caesar(X) = X + Key Mod N

أي أن تشفير رمز ما X هو باقي قسمة الرمز X بالإضافة إلى المفتاح على مجموع الرموز التي تتضمن X .

مثال رياضي :

سنعطي جدول الرموز التالي للحروف اللاتينية :

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

كما ترى فقد اخترنا فقط 26 حرفاً، إذن N = 26

قمنا بإعطاء كل حرف رقماً معيناً بالترتيب. ولكن دعنا نشفر كلمة ATRE وسنختار المفتاح 7 مثلا.

أولاً يجب أن نأخذ الحروف بشفرتها :

A = 0 ; T = 19 ; R = 17 ; E = 4

ثم نقوم بتشفير كل حرف على حدا :

Caesar(A) = 0 + 7 Mod 26 = 7 (H)
Caesar(T) = 19 + 7 Mod 26 = 0 (A)

Caesar(R) = 17 + 7 Mod 26 = 24 (Y)
Caesar(E) = 4 + 7 Mod 26 = 11 (L)

وبالتالي فإن تشفير كلمة ATRE هي HAYL ...

الأمر جميل أليس كذلك؟

إذا أردت فك التشفير فكل ما يجب عليك معرفته هو المفتاح فقط، أما الصيغة الرياضية فيمكن استنتاجها بسهولة.

صيغة فك التشفير :

X = Caesar(X) – Key Mod N

لنقم بفك كلمة HAYL :

أولاً نعطي لكل حرف قيمته :

H = 7 ; A = 0 ; Y = 24 ; L = 11

InversCeasar (H) = 7 – 7 Mod 26 = 0 (A)
InversCeasar (A) = 0 – 7 Mod 26 = - 7 Mod 26 = 45 Mod 26 = 19 (T)
InversCeasar (Y) = 24 – 7 Mod 26 = 17 (R)
InversCeasar (L) = 11 – 7 Mod 26 = 4 (E)

ملاحظة : 45 ≡ -7 [26]

والآن لم يبقَ لنا إلا استغلال هذه الخوارزمية :

قم بإنشاء مشروع جديد باسم Caesar وسنجعله KeyFileMe لأن هذا الأخير لم ينل نصيبه كالآخرين:

ثم اجعل Template كما يلي :



ثم اجعل الـ ID الـــ Unregistered Labl هو 1002

سنضع كوداً يوضح نفس المثال الذي شرحناه رياضياً وبنفس الجدول:

```
mov     eax,uMsg
.if eax==WM_INITDIALOG
    invoke CreateFile,addr RegFile,GENERIC_READ,FILE_SHARE_READ,0,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,0
    .if eax!=-1
        mov hFile,eax
        invoke ReadFile,hFile,addr RegBuffer,512,addr nBytes,0
        invoke lstrlen,addr RegBuffer
        mov Len,eax
        xor eax,eax
        xor ebx,ebx
        mov ecx,26
        nextChar:
        mov al,Byte ptr [RegBuffer+ebx]
        invoke OrginalToTable
        add al,7
        div cl
        invoke TableToOrginal
        mov Byte ptr [RegBuffer+ebx],al
        inc ebx
        cmp ebx,Len
        jnz nextChar
        invoke SetDlgItemText,hWin,1002,addr RegBuffer
        invoke CloseHandle,hFile
    .else
        invoke ExitProcess,0
    .endif
```

ثم إجرائيي OrginalToTable و TableToOrginal

```
OrginalToTable proc
xor esi,esi
mov edi,-1
mov esi,40h
next:
inc esi
inc edi
cmp eax,esi
jnz next
mov eax,edi
ret
OrginalToTable endp
TableToOrginal proc
xor esi,esi
mov edi,40h
mov esi,-1
mov al,ah
xor ah,ah
next:
inc esi
inc edi
cmp eax,esi
jnz next
mov eax,edi
ret
TableToOrginal endp
```

أما البيانات فكالتالي :

```
.data
RegFile db "AT4RE.bin",0
.data?
hFile dd ?
nBytes dd ?
RegBuffer db 20 dup(?)
Len dd ?
```

لماذا احتجنا الإجرائيتين OrginalToTable و TableToOrginal ؟

كما تعلم فإن قيم الحروف والرموز من جدول ASCII ونحن نريدها كما في الجدول، أي أننا سنعطي لـــ A القيمة 0 و لـــ B القيمة 1 . كل ما فعلته في الإجرائيتين هو خوارزمية بسيطة لتحويل القيمة الأصلية ASCII إلى قيم الجدول أعلاه والعكس أيضاً.

والآن جمع الملف ثم حاول كسره.

تجد سورس شامل للفكرة هنا تحت [7.2.1]

7.3 خوارزميات الهاش

هذا النوع من الخوارزميات هو السائد الآن للحفاظ على المعلومات بشكل محمي تقريباً 100% وأشهرها MD5 لعلك سمعت أو قرأت عنها، ولعلك أيضا تساءلت: لماذا لا يمكن عكسها مادامت منطقية وتعتمد معادلات ولا تعتمد على قيم عشوائية ؟!

الجواب : لا أدري كيف تم اختراعها لكن من وجهة نظرنا يمكننا أن نقوم بتبسيطها لك بالمثال التالي :

لنأخذ الحرف A قيمة ASCII له هي 41h الآن سنجري عدة عمليات على هذه القيمة :

41h * ABCDh = 2B9F0Dh

نأخذ فقط القيمة الوسطى 9Fh ثم نجري عليها مثالاً مشابهاً :

9Fh * F666h + F9h*123456h = 124DF100 h

ثم نأخذ منها قيمة معينة مثلاً 4Dh ونجري عملية أخرى وهكذا... وفي كل مرة نأخذ فقط قيمة من العملية :

ونحصل أخيراً على هاش بهذا الشكل :

9F4D3996D60790CD11C04F842EBC43C

الآن ماذا لو أعطيتك هذه القيمة وقلت لك استنتج لي النص الأصلي الذي أعطى هذا الهاش؟ طبعاً لن تستطيع أبدا إلا بالتخمين لماذا؟ لأنه لديك الخوارزمية فقط وليست لديك القيم المحذوفة وستضطر إلى تخمينها.

لكن لماذا نرى شرح هذه الخوارزمية في الكتب طويلاً جداً ومعقداً ؟ الجواب : لتخفيض نسبة الحصول على نفس الهاش لنفس البيانات.

كما تعلم فهاش خوارزمية MD5 يتكون من 32 حرفاً، وكل حرفٍ يمكن أن يكون 16 حرفاً مختلفاً، إذا حسب المبدأ الأساسي للتعداد فسنحصل على :

16 ^ 32 = 3,4028236692093846346337460743177e+38 هاش!! إنه رقم خيالي جداً!!

سورس الفكرة تجده هنا تحت [7.3]

برمجة Search and Replace بلغة Delphi

SearchNReplace patch coding (Delphi)

I. المقدمة

برمجة أداة تبحث عن قيم معينة في الملفات بأنواعها ثم تحول بعد ذلك تغيير هذه القيم أمر ليس بجديد، وفي أيامنا هذه يوجد الكثير من الملفات المصدرية في النت بكل اللغات تتطرق إلى عملية البحث وتبديل بايتات هنا وهناك.

و لكن أخي القارئ معرفة برمجة هذه الأداة يعتبر من أساسيات الهندسة العكسية التي يجب على المبتدئين تعلمها والإلمام بكل خباياها.

سنتطرق إن شاء الله إلى هذه العملية في هذا الدرس بشكل عـــــــــــام مع وضع الخوارزميات والإجراءات التي نحتاجها لإنجاز هذه الأداة وعملها بطريقة صحيحة.

- OllyDbg .
- Delphi 7 .
- HexWorkshop v5 للمقارنة.
- تمرين تحدي الأخ Ahmad_k مرفق.

تحميل هذه الأدوات من منتدى الفريق العربي للهندسة العكسية، قسم الأدوات.

هذا الدرس موجه إلى المستوى الشبه متوسط، وهو مدعوم بالصور ليتسنَّ لك الفهم أكثر إن شاء الله.

الملفات التي عمل عليها هذا الدرس مرفقة لكي تقوم أخي الكريم بالتجربة مع اتباع كل المراحل والوصول إلى النتيجة المرجوة.

1.1 الخوارزميات والإجراءات

يجب الحرص الشديد في برمجة أي أداة مهما كان دورها على عمل الفحوصات اللازمة التي تحتاجها الأداة، في درسنا سنتطرق إلى الفحوصات التالية :

- صحة القيم التي نبحث عنها، هل هي قيم Hexadecimal ؟
- تطابق طول القيم القديمة والجديدة.
- عمل نسخة احتياطية للملف قبل التغيير في القيم.
- التأكد من أن الملف قد تم تغييره من قبل.
- إظهار رسائل تمكن المستعمل فهم مراحل كل عملية.

1.2 صحة القيم التي نبحث عنها هل هي قيم Hexadecimal

```
function TestHexValue(TargetNumber: string; ResultLbl: TLabel; ActionEdt: TEdit;
ActionBtn: TButton): Boolean;
var i: longint;
  TmpString: string;
begin
  Result := False;

  if TargetNumber = '' then Exit;
  TmpString := TargetNumber;
  TargetNumber := '';
  for i := 1 to Length(TmpString) do
    TargetNumber := TargetNumber + Upcase(TmpString[i]);

  for i := 1 to Length(TargetNumber) do
    if ((TargetNumber[i] < '0') or (TargetNumber[i] > '9')) and ((TargetNumber[i]
< 'A') or (TargetNumber[i] > 'F'))
      then
    begin
      ActionEdt.MaxLength := 1;
      ResultLbl.Font.Color := ClRed;
      ResultLbl.Caption := 'BAD';
      ActionBtn.Enabled := False;
      Result := False;
      Exit;
    end else
    begin
      ActionEdt.MaxLength := 0;
      ResultLbl.Font.Color := ClGreen;
      ResultLbl.Caption := 'GOOD';
      ActionBtn.Enabled := True;
      Result := True;
    end;
end;
```

فائدة هذه الخوارزمية هي التحقق من المحارف المدخلة، وعدم السماح بإدخال أي محرف غير Hexadecimal وتعطيل وتفعيل زر البحث مع إظهار رسائل تعلمنا بذلك.

1.3 تطابق طول القيم القديمة والجديدة

```
if Length(OriginalValueEdt.Text) = Length(PatchedValueEdt.Text) then
 begin
   EqualityReportLbl.Caption := 'GOOD';
   EqualityReportLbl.Font.Color := ClGreen;
end else
begin
  EqualityReportLbl.Caption := 'BAD';
  EqualityReportLbl.Font.Color := ClRed;
end;
```

هذا الجزء من الإجرائية يقوم بفحص طول القيم المدخلة وإظهار رسائل تعلمنا بتطابق طولها أم لا.

1.4 عمل نسخة احتياطية للملف قبل التغيير في القيم

```
procedure BackupFile(sFileName: string; ReportMmo: TMemo);
begin
  CopyFile(Pchar(sFileName), Pchar(sFileName + '.Bak'), True);
  ReportMmo.Lines.Add('Original file backup... OK');
end;
```

هذه الإجرائية مهمة جداً لأنها تجنبنا ضياع الملف الأصلي في حالة حدوث خطأ في كتابة القيم الجديدة فيه، وهي تقوم بنسخ الملف الأصلي وإضافة امتداد Bak. إلى النسخة المخفوظة.

1.5 التأكد من أن الملف قد تم تغييره من قبل

```
  if Pos(sOriginalByte, Str) > 0 then
  begin
    …..
    …..
    …..
    …..
end
  else
  begin
    Main.ReportMmo.Lines.Add('Already Patched');
  end;;
```

جزء مستخلص من إجرائية التغيير والذي يمثل الفحص عن القيم المراد تغييرها هل هي موجودة أم لا؟

1.6 خوارزمية التحويل

طريقة البحث المعتمدة في هذا الدرس هي القيام بتحويل قيم Hexadecimal إلى Characters بخوارزمية FormatByte ثم البحث عن هذه القيم الأخيرة في الملف الهدف.

```
function FormatByte(sValue: string): string;
var i: cardinal;
  sData: Byte;

begin
  for i := 0 to (length(sValue) div 2) - 1 do
  begin
    sData := StrToInt('$' + Copy(sValue, (i * 2 + 1), 2));
    Result := Result + Char(sData);
  end;
end;
```

1.7 خوارزمية البحث

تقوم بفتح الملف الهدف بصلاحيات القراءة فيه، ثم تضع قيم الملف كاملاً في المتغير Srt ، تبدأ القراءة من أول قيمة في الملف إلى آخر قيمة، معتمدة في القراءة على حجم الملف الذي تم حفظه في المتغير i . وفي حالة وجود تطابق تظهر رسالة تعلمنا بذلك، وتتوقف على عنوان القيمة التي وجدت.

```
function Search(sFileName, sOriginalByte: string): Boolean;
var
  F: file;
  Str: string;
  I: Integer;
begin

  AssignFile(F, sFileName);
  Reset(F, 1);
  I := FileSize(F);
  SetLength(Str, FileSize(F));
  BlockRead(F, Str[1], I);

  if Pos(sOriginalByte, Str) > 0 then
  begin
    Main.ReportMmo.Lines.Add('First byte Offset : 0x' +
IntToHex(Pos(sOriginalByte, Str) - 1, 8));
    Main.ReportMmo.Lines.Add('');
    Result := True;
  end
  else
  begin
    Main.ReportMmo.Lines.Add('Nothing found...');
    Result := False;
```

```
  end;

  CloseFile(F);
end;
```

1.8 خوارزمية التغير

تقوم بفتح الملف الهدف بصلاحيات القراءة والكتابة فيه، وتقوم بنفس الفحص الخاص بخوارزمية البحث.

أعدنا كتابتها لتفادي الخطأ؛ لأن التأكد مفيد دائماً قبل التغيير في الملفات.

في حالة الحصول على تطابق تتم عملية نسخ ملف احتياطي ثم كتابة القيم الجديدة في الملف.

```
procedure Replace(sFileName, sOriginalByte, sPatchedByte: string);
var
  BytesWritten: LongWord;
  F: file;
  Str: string;
  I, Y: Integer;
begin

  AssignFile(F, sFileName);
  Reset(F, 1);
  I := FileSize(F);
  SetLength(Str, FileSize(F));
  BlockRead(F, Str[1], I);

  if Pos(sOriginalByte, Str) > 0 then
  begin
    Seek(F, Pos(sOriginalByte, Str) - 1);

    BackupFile(sFileName, Main.ReportMmo);

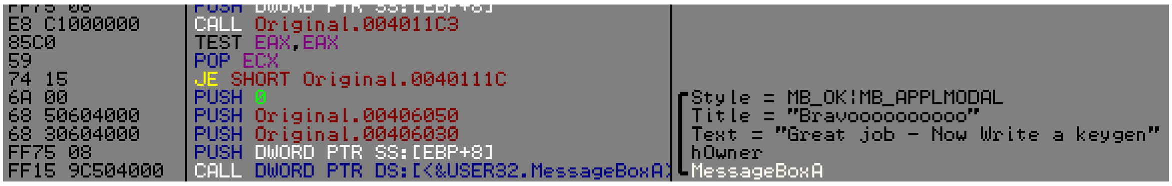
    for y := 1 to Length(sPatchedByte) do
    begin
      BlockWrite(F, sPatchedByte[y], SizeOf(sPatchedByte[y]), BytesWritten);
      if sPatchedByte[y] <> sOriginalByte[y] then
      begin
        Main.ReportMmo.Lines.Add('Original value : ' +
IntToHex(Ord(sOriginalByte[y]), 2) + ' Patched value : ' +
IntToHex(Ord(sPatchedByte[y]), 2));
      end;
    end;
  end
  else
  begin
    Main.ReportMmo.Lines.Add('Already Patched');
  end;
  CloseFile(F);
end;
```

تمت برمجة الخوارزميات والإجراءات باعتماد البساطة في الكود لتتمكن أخي القارئ من الفهم العام لعملية البحث والكتابة، ولمكنك تخصيصه حسب احتياجاتك باحترام القواعد العامة للبرمجة لتفادي الخطاء.

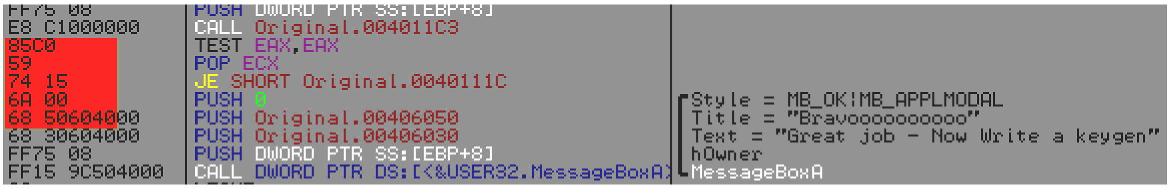
2. تطبيق عملي

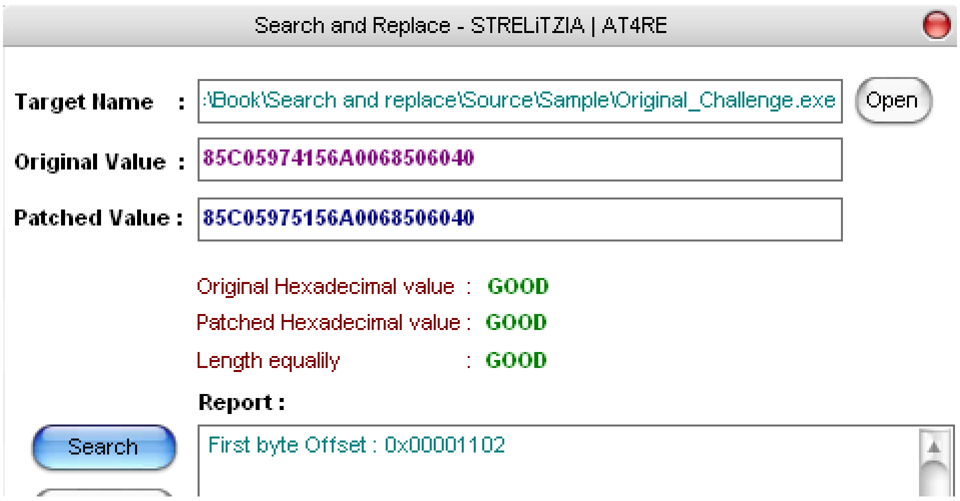
2.1 البحث

ننقح المثال المرفق تحدي الأخ Ahmad_k ثم نقوم بالتتبع حتى نصل إلى عنوان رسالة الـــ Sucess ونلاحظ الرسالة في المنقح :



ثم القيم المختارة لهذا الدرس : C05974156A006850604085



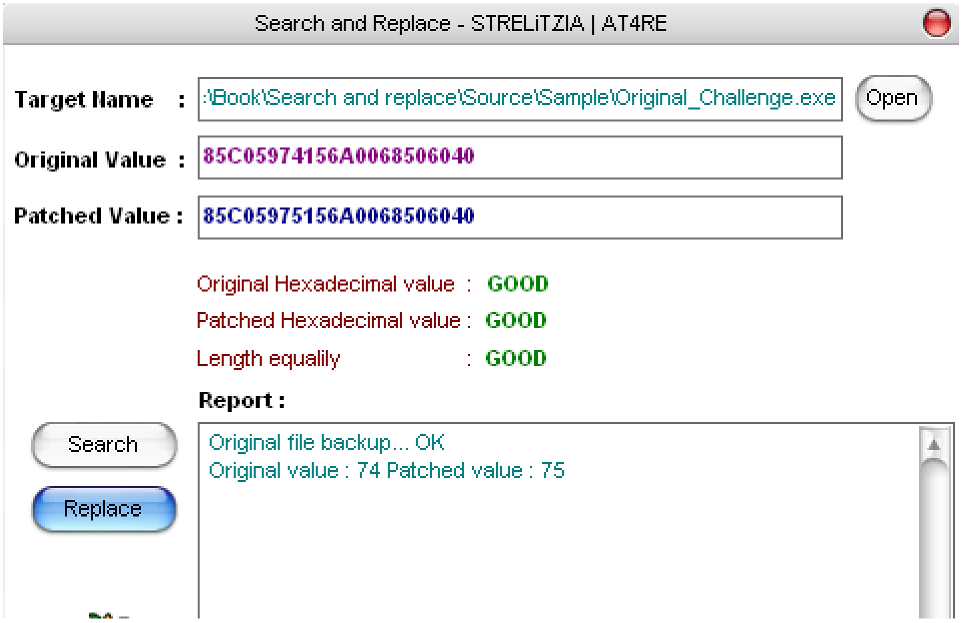


عملية البحث: تحميل الملف الهدف، إدخال القيمة المتحصل عليها من المنقح في مكان Original value، إدخال القيمة الجديدة في مكان Patched Value، مراعاة رسائل التنبيه، ثم نقر زر Search.

بعدها يظهر الناتج في Report.

2.2. التغيير

Original value	:	85C05974156A0068506040
Patched value	:	85C05975156A0068506040



عملية التغيير بسيطة ويكفي فقط نقر زر Replace لتتم العملية بنجاح.



3. الخاتمة

الحمد لله تم هذا الدرس وإن شاء الله يكون ذو منفعة وسهل الفهم.

استعمال RDTSC لكشف المنقح

Debugger Detection using RDTSC trick



1. مقدمة:

1.1 الأهداف

- كشف المنقح.
- استغلال RDTSC لخدع الـ Reverser وتوجيهه إلى خوارزمية توليد خاطئة.

1.2 تعاريف ومتفرقات

RDTSC اختصار لـ Read Time Stamp Counter وهو عبارة عن عداد يعمل داخل الـ Processor ويقرأ الزمن المنقضي بين كل Instruction.

للمزيد من الفهم الدقيق لـ RDTSC ننصحك أخي القارئ بالاطلاع على تعريفه في موسوعة Wikipedia لأن هدفنا من هذا الدرس هو تعلم استعمالها ودمجها في برنامج تجريبي، ومن ثم سوف نحاول تحليله إن شاء الله، البرنامج باسم UnderstandMe وهو مرفق مع هذا الدرس.



- OllyDbg.
- Delphi 7.

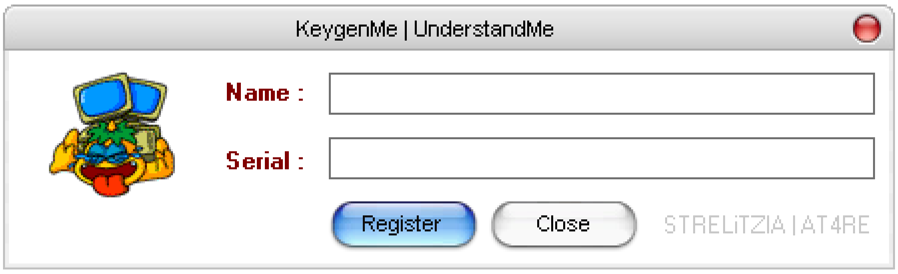


الملفات التي عمل عليها هذا الدرس مرفقة لتتمكن من التجربة مع اتباع كل المراحل والوصول إلى النتيجة المرجوة.

1.3 البداية

نشغل دلفي ثم ننشئ مشروعً جديداً ثم نحفظ كل الملفات في مجلد جديد يحمل اسم التمرين، نقوم بوضع المكونات اللازمة، وقد افترضنا أنك –أخي العزيز– تعرف قليلاً من مبادئ البرمجة بدلفي لكي لا نتطرق إليها في هذا الدرس ونخرج من الموضوع. أرفقنا مع الدرس الملفات المصدرية لتجنب الخطأ وتسهيل الفهم بالنسبة للمبتدئين.

المشروع في النهاية سيكون بهذا الشكل:



في الزر الخاص بالتسجيل Register نكتب الكود الخاص بخوارزمية معالجة اسم المستخدم لتوليد Serial ثم مقارنته بالـــ Serial المدخل، نأخذ هذا المثال :

خوارزمية تحسب قيم الاسم المدخل ثم تقوم بعمليات حسابية بسيطة للوصول إلى serial بسيط :

```
Trim(NameEdt.Text);
  sSerial := 0;
  sName := NameEdt.Text;
  if NameEdt.Text <> '' then
  begin
    for i := 0 to Length(sName) do
    begin
      sSerial := sSerial + (Ord(sName[i]) * $128);
    end;
    if a = b then
      SerialTest := IntToHex(sSerial, 2) + IntToStr(sSerial * $512) else
      SerialTest := IntToHex(sSerial, 16) + IntToStr(sSerial * $16);
```

```
    if SerialEdt.Text = SerialTest then
      ShowMessage('[Sucess] Good reverser :)') else
      ShowMessage('[Error] Bad reverser :( Try again...');
```

سنضيف كوداً نستدعي به RDTSC في حدث OnCreate يعني عند بداية تشغيل التمرين، ونحفظ الناتج في متغير عام Global نسميه "a" ثم نكتب استدعاءً في حدث OnClick التابع لزر Register ثم نحفظ الناتج في متغير عام آخر Global نسميه "b" . شرح العملية في الجزء التطبيقي :

```
var
  Main: TMain;
  a, b integer;
implementation
```

في حالة تشغيل البرنامج خارج المنقح فإن مرور الـ Instructions سيكون بسرعة كبيرة وستكون القيم الناتجة المحفوظة في المتغيرات a و b متماثلة، أما عند تشغيل التمرين داخل المنقح فإن المرور بين instructions يتحكم فيه المستعمل بسرعة مختلفة وبطيئة، مما يجعل القيم المحفوظة في المتغيرات السابقة مختلفة، وهنا يكمن الفحص وهو مقارنة بين قيم المتغيرات a و b وحسب طبيعة الناتج نوجه المستعمل إلى خوارزمية التوليد الصحيحة أو الخاطئة لنزيد الطين بله.

2. العمل التطبيقي

2.1 برمجة خوارزمية RDTSC في دلفي

```
function GetRDTSC: LongInt;
var
  TimerHi: DWord;
  TimerLo: Dword;
begin
  Sleep(10);
  asm
    dw  310Fh // rdtsc
    mov TimerLo, eax
    mov TimerHi, edx
  end;
  Sleep(500);
  asm
    dw  310Fh // rdtsc
    sub eax, TimerLo
    sbb edx, TimerHi
    mov TimerLo, eax
    mov TimerHi, edx
  end;
```

```
  Result := TimerLo div 500000;
end;
```

ناتج الخوارزمية بعد المعالجة يرجع في result وننقله نحن إلى المتغيرات السالفة الذكر: a و b.

أولاً في حدث OnCreate بداية تشغيل التمرين :

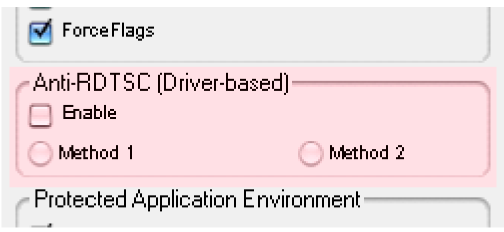
```
procedure TMain.FormCreate(Sender: TObject);
begin
   RDTSC
end;
```

ثانياً في حدث OnClick الخاص بزر التسجيل :

```
procedure TMain.RegisterBtnClick(Sender: TObject);
      …
      …
      …
begin
      RDTSC
      …
      …
      …
end;
```

2.2. التتبع في المنقح

قبل البداية يجب تعطيل مضاد RDTSC الموجود في OllyDbg Advenced Plugin كالتالي :



إلغاء تفعيل Enable الخاص بـ Anti-RDTSC (Driver-based)

بعد الانتهاء من العملية نحمل التمرين في المنقح، ثم نضع نقطة توقف عند عناوين الحدثين OnCreate و OnClick التابعين للزر Register ونبدأ التتبع...

2.2.1. حدث OnCreate

```
00453AD4  /.  55            PUSH EBP
00453AD5  |.  8BEC          MOV EBP,ESP
00453AD7  |.  83C4 F8       ADD ESP,-8
00453ADA  |.  6A 0A         PUSH 0A                          ; /Timeout = 10. ms
00453ADC  |.  E8 EB8CFBFF   CALL <JMP.&kernel32.Sleep>       ; \Sleep
00453AE1  |.  0F31          RDTSC
00453AE3  |.  8945 FC       MOV DWORD PTR SS:[EBP-4],EAX
00453AE6  |.  8955 F8       MOV DWORD PTR SS:[EBP-8],EDX
00453AE9  |.  68 F4010000   PUSH 1F4                         ; /Timeout = 500. ms
00453AEE  |.  E8 D98CFBFF   CALL <JMP.&kernel32.Sleep>       ; \Sleep
00453AF3  |.  0F31          RDTSC
00453AF5  |.  2B45 FC       SUB EAX,DWORD PTR SS:[EBP-4]
00453AF8  |.  1B55 F8       SBB EDX,DWORD PTR SS:[EBP-8]
00453AFB  |.  8945 FC       MOV DWORD PTR SS:[EBP-4],EAX
00453AFE  |.  8955 F8       MOV DWORD PTR SS:[EBP-8],EDX
00453B01  |.  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
00453B04  |.  B9 20A10700   MOV ECX,7A120
00453B09  |.  33D2          XOR EDX,EDX
00453B0B  |.  F7F1          DIV ECX
00453B0D  |.  A3 046C4500   MOV DWORD PTR DS:[456C04],EAX
00453B12  |.  59            POP ECX
00453B13  |.  59            POP ECX
00453B14  |.  5D            POP EBP
00453B15  \.  C3            RET
```

- في بداية تشغيل البرنامج نحصل على القيمة التي أرجعت بعد معالجة RDTSC في EAX ثم حفظت في المتغير a وعنوانه في المنقح هو : 00456C04

MOV DWORD PTR DS:[456C04],EAX

- نلاحظ أيضاً أن القيمة الناتجة تتغير بتغير سرعة التنقل بين كل تعليمة، وإذا وضعنا نقطة توقف في بداية الخوارزمية ثم انتقلنا إلى آخرها فسيكون الناتج دائماً مختلفاً بعد كل تتبع.
- وإذا وضعنا نقطة توقف في آخر الخوارزمية فسيكون الناتج دائماً هو نفسه لأن التنقل بين التعليمات تم بسرعة الـــ Processor.

2.2.2. حدث OnClick

```
004538C8  /.  55            PUSH EBP
004538C9  |.  8BEC          MOV EBP,ESP
004538CB  |.  B9 06000000   MOV ECX,6
004538D0  |>  6A 00         /PUSH 0
004538D2  |.  6A 00         |PUSH 0
004538D4  |.  49            |DEC ECX
004538D5  |.^ 75 F9         \JNZ SHORT Understa.004538D0
004538D7  |.  51            PUSH ECX
004538D8  |.  53            PUSH EBX
004538D9  |.  8BD8          MOV EBX,EAX
004538DB  |.  33C0          XOR EAX,EAX
004538DD  |.  55            PUSH EBP
004538DE  |.  68 713A4500   PUSH Understa.00453A71
004538E3  |.  64:FF30       PUSH DWORD PTR FS:[EAX]
004538E6  |.  64:8920       MOV DWORD PTR FS:[EAX],ESP
004538E9  |.  6A 0A         PUSH 0A                          ; /Timeout = 10. ms
004538EB  |.  E8 DC8EFBFF   CALL <JMP.&kernel32.Sleep>       ; \Sleep
004538F0  |.  0F31          RDTSC
004538F2  |.  8945 FC       MOV DWORD PTR SS:[EBP-4],EAX
004538F5  |.  8955 F8       MOV DWORD PTR SS:[EBP-8],EDX
004538F8  |.  68 F4010000   PUSH 1F4                         ; /Timeout = 500. ms
004538FD  |.  E8 CA8EFBFF   CALL <JMP.&kernel32.Sleep>       ; \Sleep
00453902  |.  0F31          RDTSC
00453904  |.  2B45 FC       SUB EAX,DWORD PTR SS:[EBP-4]
00453907  |.  1B55 F8       SBB EDX,DWORD PTR SS:[EBP-8]
0045390A  |.  8945 FC       MOV DWORD PTR SS:[EBP-4],EAX
0045390D  |.  8955 F8       MOV DWORD PTR SS:[EBP-8],EDX
00453910  |.  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
```

بعد النقر على زر Register نحصل على القيمة التي أرجعت بعد معالجة RDTSC في EAX ثم حفظت في المتغير b وعنوانه في المنقح هو : 00456C08

2.3 المقارنة بين النتائج

```
00453989  >  A1 046C4500     MOV EAX,DWORD PTR DS:[456C04]
0045398E  .  3B05 086C4500   CMP EAX,DWORD PTR DS:[456C08]
00453994  .v 75 39           JNZ SHORT Understa.004539CF
00453996  .  8B45 F4         MOV EAX,DWORD PTR SS:[EBP-C]
00453999  .  33D2            XOR EDX,EDX
0045399B  .  52              PUSH EDX                          ; Arg2 => 00000000
0045399C  .  50              PUSH EAX                          ; Arg1
0045399D  .  8D55 DC         LEA EDX,DWORD PTR SS:[EBP-24]
004539A0  .  B8 02000000     MOV EAX,2
004539A5  .  E8 7244FBFF     CALL Understa.00407E1C            ; Understa.00407E1C
004539AA  .  8B45 DC         MOV EAX,DWORD PTR SS:[EBP-24]
004539AD  .  50              PUSH EAX
004539AE  .  6945 F4 1205000 IMUL EAX,DWORD PTR SS:[EBP-C],512
004539B5  .  33D2            XOR EDX,EDX
004539B7  .  52              PUSH EDX                          ; Arg2 => 00000000
004539B8  .  50              PUSH EAX                          ; Arg1
004539B9  .  8D45 D8         LEA EAX,DWORD PTR SS:[EBP-28]
004539BC  .  E8 0B44FBFF     CALL Understa.00407DCC            ; Understa.00407DCC
004539C1  .  8B4D D8         MOV ECX,DWORD PTR SS:[EBP-28]
004539C4  .  8D45 EC         LEA EAX,DWORD PTR SS:[EBP-14]
004539C7  .  5A              POP EDX
004539C8  .  E8 8B07FBFF     CALL Understa.00404158
004539CD  .v EB 34           JMP SHORT Understa.00453A03
004539CF  >  8B45 F4         MOV EAX,DWORD PTR SS:[EBP-C]
004539D2  .  33D2            XOR EDX,EDX
004539D4  .  52              PUSH EDX                          ; Arg2 => 00000000
004539D5  .  50              PUSH EAX                          ; Arg1
004539D6  .  8D55 D4         LEA EDX,DWORD PTR SS:[EBP-2C]
004539D9  .  B8 10000000     MOV EAX,10
004539DE  .  E8 3944FBFF     CALL Understa.00407E1C            ; Understa.00407E1C
004539E3  .  8B45 D4         MOV EAX,DWORD PTR SS:[EBP-2C]
004539E6  .  50              PUSH EAX
004539E7  .  6B45 F4 16      IMUL EAX,DWORD PTR SS:[EBP-C],16
004539EB  .  33D2            XOR EDX,EDX
004539ED  .  52              PUSH EDX                          ; Arg2 => 00000000
004539EE  .  50              PUSH EAX                          ; Arg1
004539EF  .  8D45 D0         LEA EAX,DWORD PTR SS:[EBP-30]
004539F2  .  E8 D543FBFF     CALL Understa.00407DCC            ; Understa.00407DCC
004539F7  .  8B4D D0         MOV ECX,DWORD PTR SS:[EBP-30]
004539FA  .  8D45 EC         LEA EAX,DWORD PTR SS:[EBP-14]
004539FD  .  5A              POP EDX
```

في حالة تشغيل التمرين خارج المنقح تكون القيمتين المحفوظتين في المتغيرات a و b متطابقتين، ويتم الذهاب إلى خوارزمية توليد رقم التسجيل الأولى، وفي حالة عدة التطابق نذهب إلى خوارزمية توليد رقم التسجيل الثانية.

```
00453989  >  A1 046C4500     MOV EAX,DWORD PTR DS:[456C04]
0045398E  .  3B05 086C4500   CMP EAX,DWORD PTR DS:[456C08]
```

يعني الـــ Reverser يتتبع الخوارزمية ويحصل على رقم تسجيل يعمل داخل المنقح فقط وفي حالة تجربيه خارج المنقح لن يعمل عنده، وهذا هو الهدف من الخدعة.

يكمن أيضا زرع RDTSC في أمكنة كثيرة من البرنامج وحسب طبيعة الناتج يتأثر البرنامج برغبات المبرمج، من Exceptions إلى رسائل تنبيه أو غيرها.

تعمدنا في هذا الدرس تغيير جزء صغير في خوارزميات توليد رقم التسجيل وجعلها متقاربة لتتمكن أخي العزيز من الفهم بأبسط الأمثلة، ويمكنك أن تعقد الأمور في الخوارزميات المرفقة، فقط احرص على الدقة في البرمجة لإجبار الـ Reverser المرور بخوارزمية RDTSC يدوياً بـ F8 لتضمن أن الناتج سيكون مختلفاً بين بداية تشغيل البرنامج وحدث OnClick مثلاً.

> استعمال خدعة RDTSC تكشف المنقح، وتجاوز هذه الخدعة سهل ويكفي فقط تنصيب OllyDbg Advenced Plugin وتفعيل خيار Anti-RDTSC.
>
> سوف نتطرق أيضاً إلى استعمالها في درس كشف المنقح باستغلال ثغرة حماية Anti-RDTSC المقترحة في الـ OllyDbg Advenced Plugin.

2.3.1. توابع :

خارج المنقح

Name : STRELiTZIA
Serial : 3A9B8311592688

داخل المنقح

Name : STRELiTZIA
Serial : 000000000003A9B85281232

3. الخاتمة

الحمد لله تم هذا الدرس وإن شاء الله تجد فيه المتعة والفائدة.

كشف درايفر Advanced OllyDbg لكشف المنقح

Advanced OllyDbg Driver Detection | AODD Trick





الملفات التي عمل عليها هذا الدرس مرفقة ليتسنَّ لك أخي الكريم التجربة مع اتباع كل المراحل والوصول إلى النتيجة المرجوة.

I. مقدمة:

I.1 الأهداف

- كشف المنقح.

I.2 تعاريف

RDTSC اختصار لــ Read Time Stamp Counter وهو عبارة عن عداد يعمل داخل الــ Processor ويقرأ الزمن المنقضي بين كل Instruction.

للمزيد من الفهم الدقيق لــ RDTSC ننصحك أخي القارئ بالاطّلاع على تعريف موسوعة Wikipedia وكذلك الاطّلاع على الدرس السابق مع التمرين لفهم آلية RDTSC.

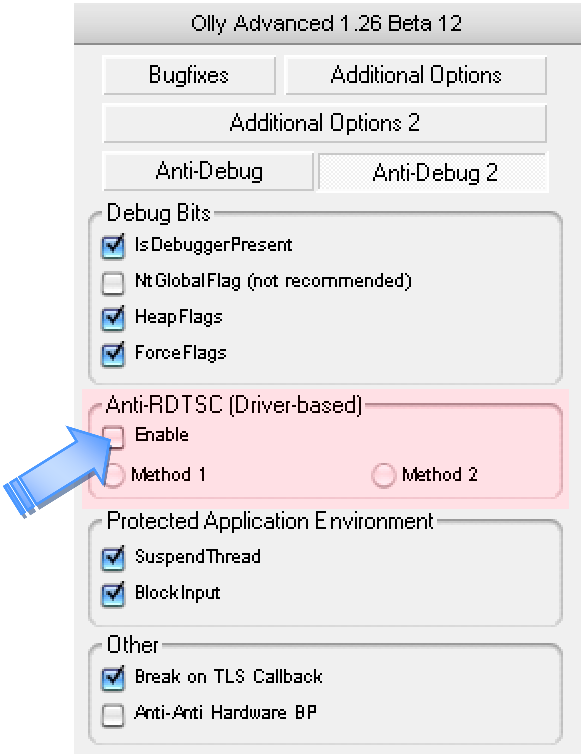
هدفنا من هذا الدرس هو كشف درايفر fakerdtsc.sys وanti_rdtsc.sys الذي اعتمد في Plugin Advenced OllyDbg لسد خدع RDTSC.

سنستعمل نفس مثال التمرين السابق UnderstandMe الذي ستجده مرفقاً مع هذا الدرس.

- OllyDbg .
- Delphi 7 .

1.3 البداية

قبل البداية يجب تفعيل خيار مضاد RDTSC الموجود في Advenced OllyDbg Plugin كالتالي :



تفعيل Enable الخاص بـ Anti-RDTSC (Driver-based)

بعد الانتهاء من العملية نحمل التمرين في المنقح، ونضع نقطة توقف عند عناوين الحدثين OnCreate و OnClick التابعين لزر Register ونبدأ التتبع...

1.3.1. حدث OnCreate

```
00453AD4  ┌.  55            PUSH EBP
00453AD5  │.  8BEC          MOV EBP,ESP
00453AD7  │.  83C4 F8       ADD ESP,-8
00453ADA  │.  6A 0A         PUSH 0A                        ┌Timeout = 10. ms
00453ADC  │.  E8 EB8CFBFF   CALL <JMP.&kernel32.Sleep>     └Sleep
00453AE1  │.  0F31          RDTSC
00453AE3  │.  8945 FC       MOV DWORD PTR SS:[EBP-4],EAX
00453AE6  │.  8955 F8       MOV DWORD PTR SS:[EBP-8],EDX
00453AE9  │.  68 F4010000   PUSH 1F4                       ┌Timeout = 500. ms
00453AEE  │.  E8 D98CFBFF   CALL <JMP.&kernel32.Sleep>     └Sleep
00453AF3  │.  0F31          RDTSC
00453AF5  │.  2B45 FC       SUB EAX,DWORD PTR SS:[EBP-4]
00453AF8  │.  1B55 F8       SBB EDX,DWORD PTR SS:[EBP-8]
00453AFB  │.  8945 FC       MOV DWORD PTR SS:[EBP-4],EAX
00453AFE  │.  8955 F8       MOV DWORD PTR SS:[EBP-8],EDX
00453B01  │.  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
00453B04  │.  B9 20A10700   MOV ECX,7A120
00453B09  │.  33D2          XOR EDX,EDX
00453B0B  │.  F7F1          DIV ECX
00453B0D  │.  A3 046C4500   MOV DWORD PTR DS:[456C04],EAX
00453B12  │.  59            POP ECX
00453B13  │.  59            POP ECX
00453B14  │.  5D            POP EBP
00453B15  └.  C3            RET
```

في بداية تشغيل البرنامج نحصل على القيمة التي أرجعت بعد معالجة RDTSC في EAX ثم حفظت في المتغير a وعنوانه في المنقح هو : 00456C04

MOV DWORD PTR DS:[456C04],EAX

نلاحظ أيضاً أن القيمة الناتجة لا تتغير بتغير سرعة التنقل بين كل تعليمة، وإذا وضعنا نقطة توقف في بداية الخوارزمية ثم انتقلنا إلى آخرها فسيكون الناتج نفسه دائماً بعد كل تتبع وهذا راجع للدريفر المخصص لإبطال RDTSC.

1.3.2. حدث OnClick

```
004538C8  ┌.  55            PUSH EBP
004538C9  │.  8BEC          MOV EBP,ESP
004538CB  │.  B9 06000000   MOV ECX,6
004538D0  │>  6A 00         ┌PUSH 0
004538D2  │.  6A 00         │PUSH 0
004538D4  │.  49            │DEC ECX
004538D5  │.^ 75 F9         └JNZ SHORT Understa.004538D0
004538D7  │.  51            PUSH ECX
004538D8  │.  53            PUSH EBX
004538D9  │.  8BD8          MOV EBX,EAX
004538DB  │.  33C0          XOR EAX,EAX
004538DD  │.  55            PUSH EBP
004538DE  │.  68 713A4500   PUSH Understa.00453A71
004538E3  │.  64:FF30       PUSH DWORD PTR FS:[EAX]
004538E6  │.  64:8920       MOV DWORD PTR FS:[EAX],ESP
004538E9  │.  6A 0A         PUSH 0A                        ┌Timeout = 10. ms
004538EB  │.  E8 DC8EFBFF   CALL <JMP.&kernel32.Sleep>     └Sleep
004538F0  │.  0F31          RDTSC
004538F2  │.  8945 FC       MOV DWORD PTR SS:[EBP-4],EAX
004538F5  │.  8955 F8       MOV DWORD PTR SS:[EBP-8],EDX
004538F8  │.  68 F4010000   PUSH 1F4                       ┌Timeout = 500. ms
004538FD  │.  E8 CA8EFBFF   CALL <JMP.&kernel32.Sleep>     └Sleep
00453902  │.  0F31          RDTSC
00453904  │.  2B45 FC       SUB EAX,DWORD PTR SS:[EBP-4]
00453907  │.  1B55 F8       SBB EDX,DWORD PTR SS:[EBP-8]
0045390A  │.  8945 FC       MOV DWORD PTR SS:[EBP-4],EAX
0045390D  │.  8955 F8       MOV DWORD PTR SS:[EBP-8],EDX
00453910  │.  8B45 FC       MOV EAX,DWORD PTR SS:[EBP-4]
00453913  │.  B9 20A10700   MOV ECX,7A120
00453918  │.  33D2          XOR EDX,EDX
0045391A  │.  F7F1          DIV ECX
0045391C  │.  A3 086C4500   MOV DWORD PTR DS:[456C08],EAX
```

بعد النقر على زر Register نحصل على القيمة التي أرجعت بعد معالجة RDTSC في EAX ثم حفظت في المتغير b وعنوانه في المنقح هو : 00456C08

الدرايفر استطاع أن يعطل الخدعة، وهذا أمر طبيعي لأنه صمم لهذا الغرض، لذا سنحاول برمجة كود يكشف الدرايفر، ويبقى فقط دمجه مع كود كشف RDTSC.

1.3.3. مراحل العملية

كشف المنقح باستعمال : RDTSC تم التطرق إليه في الدرس السابق.

كشف المنقح عن طريق كشف درايفر fakerdtsc.sys و anti_rdtsc.sys

2. البرمجة

```
program AdvancedOllyDbg;
uses
  windows, SysUtils;
const
  DRIVER_INFORMATION = 11;
  fakerdtsc = 'fakerdtsc.sys';
  anti_rdtsc = 'anti_rdtsc.sys';
type
  TPDWord = ^DWORD;
  TDriverInfo = packed record
    Address: Pointer;
    Size: DWORD;
    Unknown2: DWORD;
    EntryIndex: DWORD;
    Unknown4: DWORD;
    Name: array[0..MAX_PATH + 3] of Char;
  end;
var
  ZwQuerySystemInformation: function(infoClass: DWORD;
    buffer: Pointer;
    bufSize: DWORD;
    returnSize: TPDword): DWORD; stdcall = nil;
procedure GetDriverInfo;
var
  temp, Index, numBytes, numEntries: DWORD;
  buf: TPDword;
  driverInfo: ^TDriverInfo;
  Result: string;
begin
  if @ZwQuerySystemInformation = nil then
    ZwQuerySystemInformation := GetProcAddress(GetModuleHandle('ntdll.dll'),
'ZwQuerySystemInformation');
  ZwQuerySystemInformation(DRIVER_INFORMATION, @temp, 0, @numBytes);
  buf := AllocMem(numBytes * 2);
```

```
  ZwQuerySystemInformation(DRIVER_INFORMATION, buf, numBytes * 2, @numBytes);
  numEntries := buf^;
  driverInfo := Pointer(DWORD(buf) + 12);
  for Index := 1 to numEntries do
  begin
    Result := ExtractFileName(driverInfo^.Name);
   (* ______________________________________ *)
    if (Result = fakerdtsc) or (Result = anti_rdtsc) then
      MessageBoxA(0, '[Sucess] OllyDbg Detected'#13''#13'STRELiTZIA |
AT4RE'#13'www.at4re.com'#13'OllyDbg Advanced Anti-RDTSC Driver Detection',
'Sucess', MB_OK);
   (* ______________________________________ *)

    Inc(driverInfo);
  end;
  Delete(Result, 1, 2);
  FreeMem(buf);
end;
begin
  GetDriverInfo;
end.
```

للحصول على قائمة الدرايفر النشطة في النظام نستعمل دالة ZwQuerySystemInformation الموجودة في المكتبة الديناميكية ntdll.dll التي يتم استدعاؤها.

و هذا جزء صغير من تعريفها في MSDN Library :

NTSTATUS WINAPI ZwQuerySystemInformation(	
__in SYSTEM_INFORMATION_CLASS SystemInformationClass,	مقبض الرافذة التي سنضيف لها المؤقت
__in_out PVOID SystemInformation,	لن نحتاجه؛ لذا أعطه القيمة 0
__in ULONG SystemInformationLength,	هذا هو المهم وهو مدة المؤقت وتحسب بأجزاء الألف للثانية
__out_opt PULONG ReturnLength	لا حاجة لنا به؛ لذا أعطه القيمة 0
);	

Remarks

The ZwQuerySystemInformation function and the structures that it returns are internal to the operating system and subject to change from one release of Windows to another. To maintain the compatibility of your application, it is better to use the alternate functions previously mentioned instead.

If you do use ZwQuerySystemInformation, access the function through run-time dynamic linking. This gives your code an opportunity to respond gracefully if the function has been changed or removed from the operating system. Signature changes, however, may not be detectable.

This function has no associated import library. You must use the LoadLibrary and GetProcAddress functions to dynamically link to Ntdll.dll.

يمكنك الاطلاع على المعلومات الدقيقة الخاصة بهذه الدالة في مكتبة MSDN.

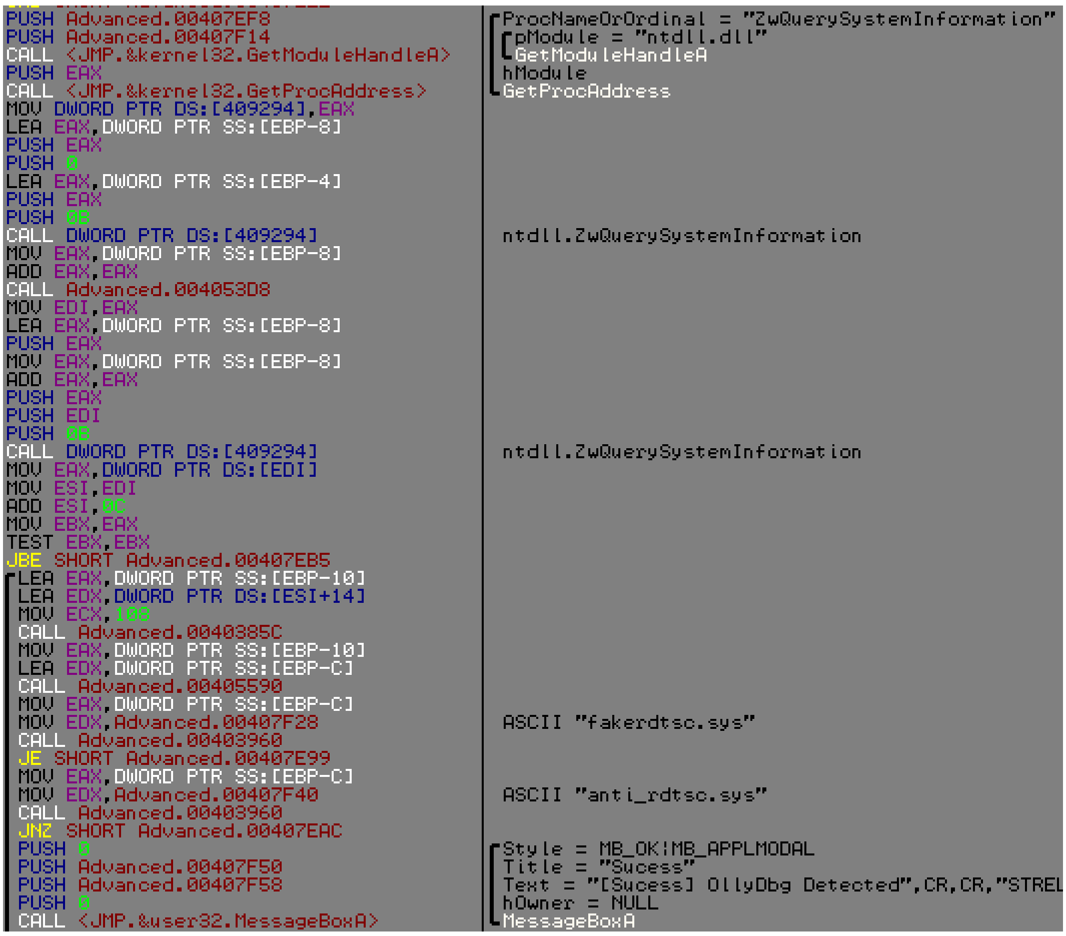
بعد إحصاء الدرايفر النشطة في النظام نقوم بمقارنة أسماء الدرايفر بـــ anti_rdtsc.sys و fakerdtsc.sys وفي حالة التطابق تظهر رسالة تعلمنا بذلك :

```
if (Result = fakerdtsc) or (Result = anti_rdtsc) then
      MessageBoxA(0, '[Sucess] OllyDbg Detected'#13''#13'STRELiTZIA |
AT4RE'#13'www.at4re.com'#13'OllyDbg Advanced Anti-RDTSC Driver Detection',
'Sucess', MB_OK);
```

في المثال المرفق تعمدنا عدم تعقيد عملية الفحص، ولكن اعلم أخي القارئ أن تخطيها سيكون سهلاً وفي متناول الجميع؛ لذا ينصح بإضافة تشفير لاسم الدرايفر ودالة الفحص وإضافة خدع أخرى تجعل التنقيح صعباً ومملاً.

3. تنقيح التمرين

نضع نقطة توقف على العنوان التالي 00407DE0 ثم نقوم بالتتبع نقطةً نقطةً لفهم ما يجري :



نلاحظ فحص كل أسماء الدريفرات النشطة في النظام ومقارنتها بالدرايفرين المذكورين سابقاً، وفي حال تطابق إحداهما تظهر رسالة تعلمنا بذلك.



يمكنك أخي القارئ دمج الخدعتين معاً، خدعة الدرس السابق RDTSC و Advanced OllyDbg Driver Detection التي سنطلق عليها إن شاء الله اسم خدعة AODD لكشف المنقح، وكل الحقوق محفوظة للفريق العربي للهندسة العكسية.

9. الخاتمة

الحمد لله تم هذا الدرس وإن شاء الله تجد فيه المتعة والفائدة.

كيفية برمجة Unpacker بسيط

Coding unpackers using Apox SDK

1. المقدمة

سنقوم في هذا الدرس بالتعرف على طريقة بسيطة لبرمجة Unpacker بلغة الدلفي وذلك بمساعدة مكتبة SDK وهي ap0x unpack engine وهي عبارة عن 4 ملفات Dll يتم إضافتهم إلى المشروع، وهذه الملفات هي :

- Debugger.dll
- Importer.dll
- Dumper.dll
- REALIGN.DLL

يمكنك تحميل هذه المكتبة من الموقع الأصلي أو ستجدها مرفقة هنا مع الدرس

وبما أن هذا الدرس هو درس برمجي لذلك سوف نقوم بوضع قطع الكود وشرحها، وسيكون التطبيق على حماية الـــ NsPack v3.x .

2. أساسيات يجب معرفتها

الخطة العامة التي سنتبعها لفك أي حماية هي :

1) الوصول إلى الـــ OEP JMP وهي القفزة الأخيرة التي تصل بنا إلى OEP أي بداية البرنامج الفعلية.

2) وضع نقطة توقف على الـــ OEP JMP .

3) تشغيل الـــ Debugger وبعد أن يتوقف على نقطة التوقف هذه نقوم بالحصول على الـــ OEP كما سترى فيما بعد.

4) باقي الخطوات لا علاقة لنا بها؛ لأنه بمجرد حصولنا على الـــ OEP ينتهي عملنا وتقوم المكتبات بباقي العمليات Dump ثم البحث عن الـــ IAT ثم إصلاح الـــ IAT ثم إعادة ترتيب الملف، وبعد ذلك حفظ النسخة النهائية.

2.1 الوصول إلى الـ OEP JMP

الملف الذي سيطبق علية الشرح هو ملف كيجن لشركة Apex وهو مضغوط بواسطة Nspack .

قم بفتح الملف في OllyDbg :

```
00496CB4   9C              PUSHFD
00496CB5   60              PUSHAD
00496CB6   E8 00000000     CALL keygen.00496CBB
00496CBB   5D              POP EBP
00496CBC   83ED 07         SUB EBP,7
00496CBF   8D8D 24FCFFFF   LEA ECX,DWORD PTR SS:[EBP-3DC]
00496CC5   8039 01         CMP BYTE PTR DS:[ECX],1
```

سنقوم بالتتبع بواسطة F8 حتى ندخل في أول loop ثم ثاني loop ثم سنصل إلى أمر JMP عند العنوان 00496F26

```
00496F1C   B8 01000000     MOV EAX,1
00496F21   C2 0C00         RETN 0C
00496F24   61              POPAD
00496F25   9D              POPFD
00496F26 - E9 4D56FCFF     JMP keygen.0045C578
00496F2B   8BB5 B0FBFFFF   MOV ESI,DWORD PTR SS:[EBP-450]
00496F31   0BF6            OR ESI,ESI
00496F33 v 0F84 97000000   JE keygen.00496FD0
00496F39   8B95 B8FBFFFF   MOV EDX,DWORD PTR SS:[EBP-448]
```

إذا ضغطت على F8 ستجد نفسك عند الـــ OEP في العنوان 0045C578 ، إذن: هذه الـــ JMP هي الـــ OEP JMP

ما علينا فعلة برمجياً هو الوصول إلى هذه القفزة. من الممكن طبعاً الوصول إليها من خلال العنوان، لكن تذكر أننا نبرمج Unpacker أي أننا لن نتعامل مع هذا الملف فقط بل مع أي ملف محمي بهذه الحماية، وكل ملف سيختلف فيه العنوان عن الملف الآخر؛ لذلك فنحن سنبحث برمجياً عن البايتات 61 9D E9 لأنها ستكون ثابتة في كل البرامج المحمية بـــ Nspack.

2.2 وضع نقطة توقف على الـ OEP JMP

سنلقي نظرة على الكود المستخدم في الـــ Unpacker ، طبعاً ستجد السورس كود مع مرفقات الكتاب.

البداية عند الإجراء :

```
procedure TForm1.BitBtn2Click(Sender: TObject);
```

وهو الإجراء الذي ينفذ عند الضغط على زر Unpack سنقوم الآن بشرح هذا الإجراء، ومن خلاله سنصل للـــ OEP JMP ونضع عليها نقطة توقف ثم نشغل الديبغر.

بداية سوف نقوم بإضافة البايتات التي سنبحث عنها، وكما ذكرنا سابقاً سوف نبحث عن 61 9D E9

```
procedure TForm1.BitBtn2Click(Sender: TObject);
 var
 NSPSignature1  :array[1..10] of BYTE;
 WildCard : Integer;
 FoundPattern,SearchSize:Integer;
begin
 if FileExists(Edit1.Text) then begin
      ImageBase := 0;
      FoundPattern := 0;
      WildCard := 0;

      NSPSignature1[1] := $61;     //NsPack signature
      NSPSignature1[2] := $9D;
      NSPSignature1[3] := $E9;
```

بعد ذلك يوجد جزئية خاصة بالحصول على الـــ ImageBase للبرنامج و Packed_OEP وهي ليست هامة سنستخدمها مع دالة InitDebug لذلك هي ثابتة مع كل البرامج، ونحن نشرح فقط الجزئيات المتغيرة.

الآن جزئية البحث عن البايتات ووضع نقطة توقف على الـــ OEP JMP ثم تشغيل الديبغر:

```
//initialize the debugging by calling the InitDebug API
      ProcessInfo :=
InitDebug(PChar(Edit1.Text),'',PChar(ExtractFilePath(Edit1.Text)));
      ListBox1.Items.Add('Creating process...');
      ListBox1.Items.Add('Searching for NsPack pattern...');


     FoundPattern := Find(PackedOEP,1000,@NSPSignature1,3,@WildCard);

     if FoundPattern > 0 then begin
//In this suspended state call SetBpx to set the main
//breakpoint at OEP
         SetBpx(FoundPattern+2,bpxSingle,@OEP_JUMP);
         ListBox1.Items.Add('Setting BPX at: ' + IntToHex(FoundPattern+2,8));
         ListBox1.Items.Add(' -> Unpacking NsPack v3.x');
//To get the debugging process to this point you
//must call the DebugLoop API
         DebugLoop();
       end

       else begin
          ListBox1.Items.Add('File is not packed with NsPack...');
          StopDebug();
          DebugLoop();
          ListBox1.Items.Add('Unpacking terminated...');
     end;
```

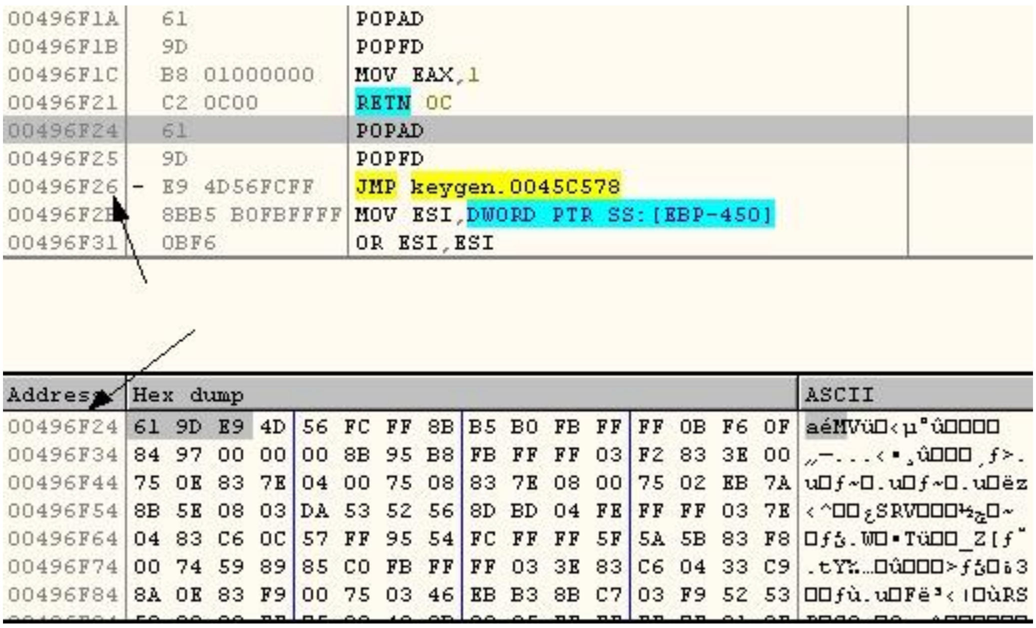
```
 end;
end;
```

1. أول دالة هي InitDebug وهي بالضبط كفتحنا البرامج بالـ OllyDbg وهي تجعل الـ Debugger في حالة انتظار suspended state .

2. هذه الدالة وظيفتها ظاهرة من اسمها، وهي للبحث عن البايتات الخاصة بال NsPack وكما ترى فإن أول باراميتر لها هو نقطة البداية الـ EP ثم بعد ذلك مجال البحث ثم البايتات التي سنبحث عنها ثم عددهم، فإذا وجدت الدالة البايتات فسترجع القيمة 1 أما إن لم تجدهم فسترجع القيمة 0 وحينها سيتم تنفيذ كود الخروج عند الـ else .

3. هذه الدالة من أهم الدوال ووظيفتها ظاهرة من اسمها، وهي خاصة بوضع نقطة التوقف وبارامتراتها كالتالي :

- أول باراميتر هو العنوان الذي سنضع علية نقطة التوقف، وفي حالتنا هو عنوان الـ OEP JMP والذي يساوي 00496F26 ولكن لماذا أضفنا 2 ؟ الجواب: لأن قيمة الـ FoundPattern تساوي 00496F24 لذلك نضيف 2 حتى تساوي عنوان الـ JMP



- ثاني باراميتر خاص بنوع نقطة التوقف.

- ثالث باراميتر هذا هام ويسمى الـ callback فكما وضحنا سابقا: كأننا حملنا البرنامج فقط في الـ OllyDbg وعندما نقوم بتشغيل البرنامج سوف يتوقف عند نقطة التوقف هذه التي وضعناها، هذا الباراميتر يحتوي على الإجرائية التي سيتم تنفيذها بعد وقوف الـ Debugger على نقطة التوقف. وهذا الإجراء هو الخاص بالوصول إلى الـ OEP وسيتم شرحه لاحقاً إن شاء الله.

4. هذه الدالة هي دالة تشغيل الديبغر، وسيقوم الديبغر بالعمل حتى يتوقف عند نقطة التوقف التي وضعناها، بعدها يتم تنفيذ الـ callback كأننا بالضبط ضغطنا على F9 بعد وضعنا لنقطة التوقف.

2.3 الحصول على الـ OEP

```
OEP := GetContextData(rEIP);
OEP := OEP + 1;
hProcess := ProcessInfo^;
ReadOk := ReadProcessMemoryFixed(hProcess,OEP,@buffer,4,@NumberOfBytesRead);

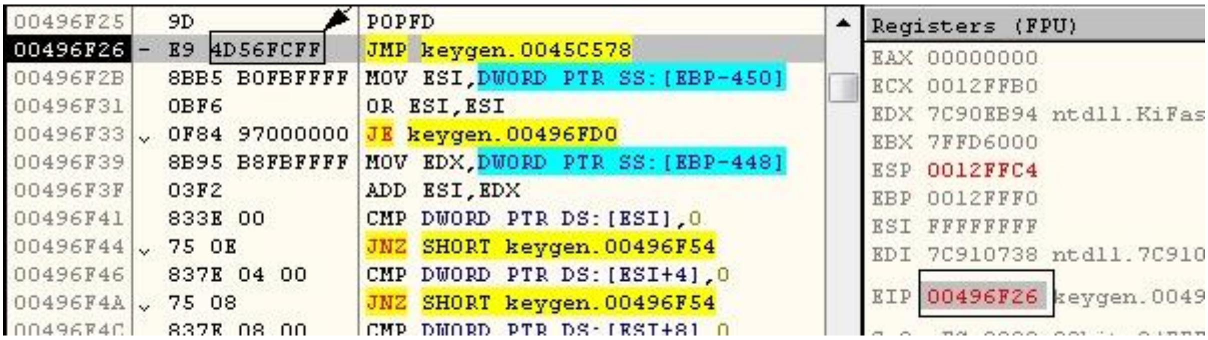
if ReadOk = True then begin
 OEP := OEP + buffer + 4;
 Form1.ListBox1.Items.Add(' -> OEP is: ' + IntToHex(OEP,8));

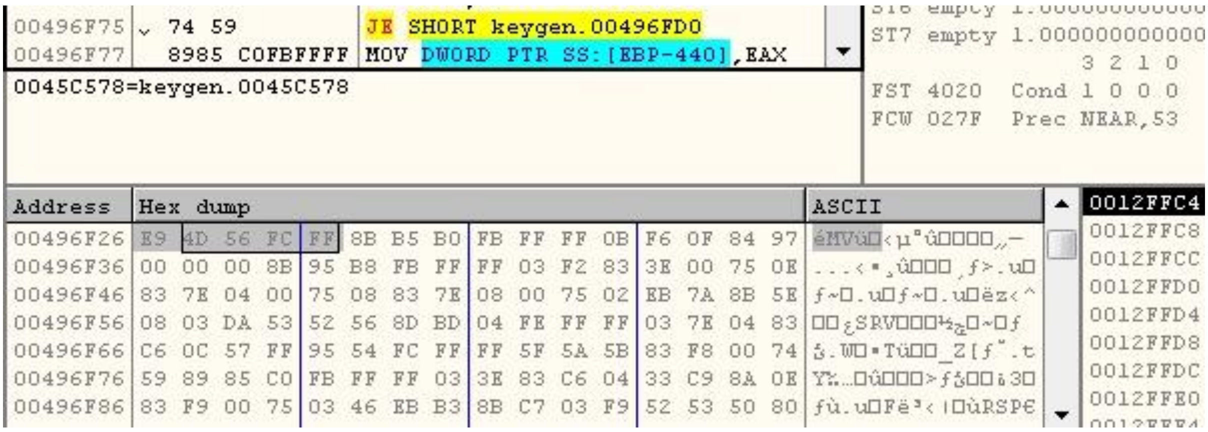
 //dump the debugged process with DumpProcess
 DumpProcess(hProcess,ImageBase,PChar(DumpTo),OEP);
 Form1.ListBox1.Items.Add(' -> Dumping process');

 //ImpREC

ImporterAutoSearchIAT(PChar(DumpTo),ImageBase,FirstSectionVA,NTSizeOfImage,@IATSt
art,@IATSize);
 Form1.ListBox1.Items.Add(' -> Searching for IAT');
 ImporterAutoFixIAT(hProcess,PChar(DumpTo),ImageBase,IATStart,IATSize,1);
 Form1.ListBox1.Items.Add(' -> Fixing IAT');
```

أول شي نقوم بقراءة محتوى الريجستري EIP بواسطة الدالة GetContextData وفي حالتنا سيكون محتواها عنوان الـ OEP JMP والذي يساوي 00496F26 لكن كيف سنحسب عنوان الـ OEP ونخرجه من خلال قراءة الـ JMP ؟ شاهد هذه الصورة:





بشكل عام يتم معرفة المكان الذي سنقفز إليه والذي يسمى الــ Jmp Destination بواسطة هذا القانون :

Destination = JMP Offset + Program Counter + 4

حيث أن JMP Offset هي القيمة FFFC564D ، لاحظ أننا عكسناها؛ لذلك فنحن نضيف 1 على الــ OEP كما في الكود في الأعلى، وذلك لكي نقرأ الأوفست عند العنوان 00496F27 ، وتتم القراءة بواسطة الدالة : ReadProcessMemoryFixed

أما الــ program Counter فهو العنوان الحالي، أي عنوان الــ OEP JMP وهذا حصلنا عليه من قراءة الــ EIP وطبعاً الــ 4 هذا ثابت.

الآن إذا طبقنا هذا القانون :

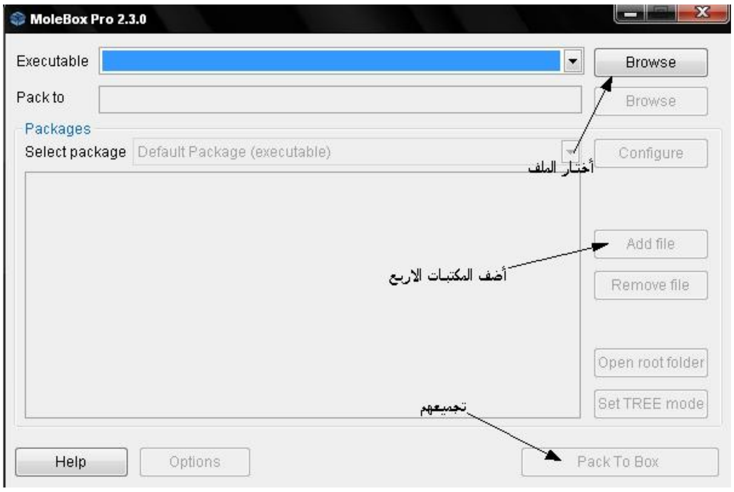
Destination = FFFC564D + 00496F26 + 4 = 10045C574 + 4 = 10045C578

وحسب طول الريجستري فسيكون الناتج 0045C578 وهو عنوان الــ OEP

بعد حصولنا على الــ OEP ينتهي عملنا ونترك الباقي على المكتبات، فيقوم البرنامج بعمل Dump للملف عن طريق الدالة DumpProcess ثم البحث عن الــ IAT عن طريق الدالة ImporterAutoSearchIAT ثم إصلاح الــ IAT عن طريق الدالة ImporterAutoFixIAT ثم عمل Rebuild للملف بواسطة مكتبة REALIGN.DLL وكل هذه الخطوات ثابتة مع أي حماية تريد عمل Unpacker لها.

2.9. تجميع ملفات الــ DLL داخل الــ Unpacker

أخيراً لتقوم بتجميع ملفات المكتبات داخل الــ Unpacker يوجد حلين: حل برمجي وحل بواسطة برنامج MoleBox وهو يقوم بجمع المكتبات داخل الــ Unpacker ويخرج لنا ملفاً نهائياً مدمج بداخله المكتبات.





يمكنك تحميل برنامج الــ MoleBox من المنتدى أو من خلال البحث في جوجل.
وستجد مع السورس كود الملف النهائي داخل مجلد Release.

Handling Debug Registers

درس اليوم عن كيفية كتابة Debugger للتحكم في سير البرنامج وكيفية وضع نقط توقف Hardware Breakpoints

تستطيع تحميل المرفقات من هنا.

I. مقدمة

كما نعلم فإن أقصى عدد للبايتات التي يمكن وضعها في الـ (Hardware Breakpoints اختصاراً لـ HWBP) هو أربع بايتات، ويتم التعامل مع نقاط التوقف عبر مسجلات التنقيح Debug Registers وهي ثمانية مسجلات، طول كل واحدٍ منهم 32 bit ووظيفتهم كالتالي :

Dr0-Dr1-Dr2-Dr3: يمثل كل مسجل منهم العنوان الذي سنطع عليه نقطة التوقف.

Dr4-Dr5 : محجوزان ولا يمكن استعمالهم.

Dr6: يسمى أيضاً Debug Status Register ويُمكننا من معرفة أي HWBP حدث. وسنتطرق له بالتفصيل لاحقاً.

Dr7: يسمى بـ Debug Control Register وهو الذي يتحكم بالـ HWBP) الكل في الكل)، وتركيبته كالتالي:

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
LEN 3	R/W 3	LEN 2	R/W 2	LEN 1	R/W 1	LEN 0	R/W 0	0	0	GD	0	0	1	GE	LE	G3	L3	G2	L2	G1	L1	G0	L0	DR7

أول 8 بتات (7 <- 0) يمكن تقسيمهم إلى اثنين اثنين، بحيث كل اثنين يخصان إحدى نقط التوقف، وإذا تم وضع أول بت (L) فإن نقطة التوقف تكون local breakpoint أي يحدث الـ Debug exception في الـ Process صاحب الـ thread الذي غيرنا Dr7 له.

ثاني بت (G) لوصف نقطة التوقف على أنها global breakpoint أي يحدث الـ debug exception إذا توافرت شروطه في جميع الـ processes.

من R/W0 إلى: R/W3

لتحديد نوع نقطة التوقف المطبقة على العنوان المحدد, فيما إذا كانت عند قراءته أو الكتابة فيه أو تنفيذه كتعليمة, يمكن وضع قيمته كالتالي (بالبت:)

00 : توقف عند تنفيذ التعليمة التي بالعنوان.

01 : توقف عند الكتابة على العنوان.

10 : لا تستعمل إلا في ظروف خاصة (للهاردوير).

11 : توقف عند القراءة أو الكتابة من العنوان.

من LEN0 إلى: LEN3

لتحديد طول البيانات التي ستضع عليها نقطة التوقف يمكن وضع قيمته كالتالي (بالبت:)

00 : لتحديد طول 1 بايت.

01 : لتحديد طول 2 بايت.

10 : تستخدم في بعض المعالجات لتحديد طول 8 بايت

11 : لتحديد طول 4 بايت.

للمزيد من التفاصيل اقرأ الـ documentations الخاصة بـ INTEL أو (AMD)

2. التطبيق العملي

حسناً لقد تعرفنا بما فيه الكفاية على مسجلات التنقيح، والآن ما الخطة التي سنعمل عليها ؟

- كيف سنقوم بوضع HWBP عملياً على Process معين ؟
- كيف سنعرف أنه تم الوصول إلى نقطة توقف معينة ؟

لتوضيح الفكرة بشكل عملي سنقوم بكتابة debugger صغير يقوم بعمل debug على برنامج crackme كمثال ووضع نقط توقف فيه وعمل أشياء أخرى.

لتنقيح برنامج معين يمكن فعل ذلك عن طريق إنشاء العملية بالدالة CreateProcess ووضع بارامتر الأعلام flags == DEBUG_PROCESS ..

كفكرة عامة ما الذي يحدث عند وقوع خطأ في برنامج معين ليس تحت التنقيح ؟

يتم إظهار الـ exception أو كما يقال exception raise: ويتم تسليم أول SEH handler التنفيذ (ليس بدقّة)، لكن ما يحدث عند وجود البرنامج تحت التنقيح أن الـ debugger هو الذي يستلم الـ execution ونوع الـ exception الذي حدث ويقرر إذا كان سيتعامل مع هذا الـ exceptionأم يجعل البرنامج نفسه (debuggee) هو الذي يتعامل مع الـ exception. نرجو أن تكون الفكرة واضحة.

دوال win32api توفر عدة دوال للتعامل مع التنقيح مثل:

WaitForDebugEvent - GetThreadContext - SetThreadContext - Read/WriteProcessMemory

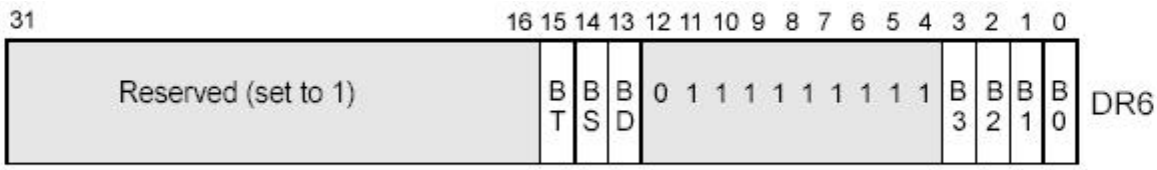
طبعا نحن هنا لسنا بصدد التعرف على تلك الدوال، يمكنك الاطلاع عليهم أكثر في مكتبة MSDN.

عند حدوث HWBP يتم رفع exception من النوع EXCEPTION_SINGLE_STEP إلى المنقح إذا كان موجوداً. إذن لابد علينا من عمل handling لهذا الـ exception code

سؤال آخر: كيف نعرف أي HWBP من الأربعة هو الذي حدث ؟

الجواب: لا توجد دوال api معينة لمعرفة ذلك!

نحن لم نتكلم عن Dr6 بالتفصيل بعد كما وعدنا وها قد حان الوقت. تركيبة هذا المسجل بسيطة جداً، كالتالي:



ما يهمنا معرفته هو أول 4 bits وهي تمثل بت لكل HWBP ، وإذا تم وضع أحدهم فهو يدل على أن الـــ HWBP المناظر له هو الذي حدث.

2.1 كيف نضع HWBP ؟

ستقول: من مسجلات التنقيح Drx ؟! لكن أنت تعلم أن استخدامها privileged ولا يمكن تغييرها من!! user-mode

سنقول لك: لا !! إذن أين ذهبت دالة SetThreadContext ؟

BOOL WINAPI SetThreadContext(	
__in HANDLE hThread,	مقبض handle للثريد الذي سنغير الـ context له
__in const CONTEXT* lpContext	: lpContextمؤشر إلى structure به الـ contextالجديد
);	

2.2 ما هو الـ Thread Context ؟

تعلمون أن نظام ويندوز يتمتع بخاصية الـــ Multithreading أي إمكانية عمل أكثر من thread في نفس الـــ process . كل threadله مسجلاته الخاصة والمكدس وأشياء أخرى.

يوجد في قلب النظام ما يسمى بالـــ Kernel Dispatcher وهو المسؤول عن تنظيم الـــ threads أو ThreadScheduling .

عند عمل swap-out لـــ threadلابد من حفظ كل ما يتعلق به (مسجلات – مكدس .. storage areas - الخ) في مكان آمن حتى إذا تم عمل swap-in يتم استرجاع البيانات من هذا المكان مرة أخرى والعكس . هذا المكان أو الذاكرة هي ما نقصده بالـــ Thread Context أو Context مباشرة.

تركيب الـــ CONTEXT strcuture يكون كالتالي:

```
struct {
    DWORD ContextFlags;

    //
```

```
    // This section is specified/returned if CONTEXT_DEBUG_REGISTERS is
    // set in ContextFlags.  Note that CONTEXT_DEBUG_REGISTERS is NOT
    // included in CONTEXT_FULL.
    //
    DWORD   Dr0;
    DWORD   Dr1;
    DWORD   Dr2;
    DWORD   Dr3;
    DWORD   Dr6;
    DWORD   Dr7;

    //
    // This section is specified/returned if the
    // ContextFlags word contians the flag CONTEXT_FLOATING_POINT.
    //
    FLOATING_SAVE_AREA FloatSave;

    //
    // This section is specified/returned if the
    // ContextFlags word contians the flag CONTEXT_SEGMENTS.
    //
    DWORD   SegGs;
    DWORD   SegFs;
    DWORD   SegEs;
    DWORD   SegDs;

    //
    // This section is specified/returned if the
    // ContextFlags word contians the flag CONTEXT_INTEGER.
    //
    DWORD   Edi;
    DWORD   Esi;
    DWORD   Ebx;
    DWORD   Edx;
    DWORD   Ecx;
    DWORD   Eax;

    //
    // This section is specified/returned if the
    // ContextFlags word contians the flag CONTEXT_CONTROL.
    //
    DWORD   Ebp;
    DWORD   Eip;
    DWORD   SegCs;              // MUST BE SANITIZED
    DWORD   EFlags;             // MUST BE SANITIZED
    DWORD   Esp;
    DWORD   SegSs;

    //
    // This section is specified/returned if the ContextFlags word
    // contains the flag CONTEXT_EXTENDED_REGISTERS.
    // The format and contexts are processor specific
    //
    BYTE    ExtendedRegisters[MAXIMUM_SUPPORTED_EXTENSION];
} CONTEXT;
```

كما ترون توجد جميع مسجلات التنقيح داخل الـــ strcuture

ما سنفعله هو كالتالي:

- جلب الـ context عن طريق الدالة GetThreadContext.
- تعديل مسجل Dr7 ووضع البتات الخاصة بطول الـــ HWBP ونوعها.
- وضع عنوان الــ HWBP في أحد المسجلات الأربعة Dr0 -> Dr3.
- إرجاع الــ context المعدلة عن طريق الدالة SetThreadContext.

هذا كود عملي مبسط لشرح الكلام النظري السابق:

```
bool SetHWBP(HANDLE hThread, unsigned int linearAddress, int type, int length,
int count)
{
    CONTEXT context = {CONTEXT_ALL|CONTEXT_DEBUG_REGISTERS};
    DR7 dr7;

    if (GetThreadContext(hThread, &context))
    {
        dr7 = *(DR7*)&context.Dr7;
        switch (count)
        {
        case 0:
            {
                context.Dr0 = linearAddress;
                dr7.HWBP0_MODE   = HWBP_LOCAL;
                dr7.HWBP0_LENGTH = length;
                dr7.HWBP0_ACCESS = type;
                break;
            };
        case 1:
            {
                context.Dr1 = linearAddress;
                dr7.HWBP1_MODE   = HWBP_LOCAL;
                dr7.HWBP1_LENGTH = length;
                dr7.HWBP1_ACCESS = type;
                break;
            };
        case 2:
            {
                context.Dr2 = linearAddress;
                dr7.HWBP2_MODE   = HWBP_LOCAL;
                dr7.HWBP2_LENGTH = length;
                dr7.HWBP2_ACCESS = type;
                break;
            };
        case 3:
            {
                context.Dr3 = linearAddress;
                dr7.HWBP3_MODE   = HWBP_LOCAL;
                dr7.HWBP3_LENGTH = length;
                dr7.HWBP3_ACCESS = type;
                break;
            };
        default:
            return false;
```

```
        };
        context.Dr7 = *(PDWORD)&dr7;
        return SetThreadContext(hThread, &context);
    };
    return false;
};
```

ولحذف HWBP نفعل كما سبق لكن نصفّر الـ bit الخاص بالـ local والمسجل الخاص بالعنوان له:

```
bool RemoveHWBP(HANDLE hThread, int count)
{
    CONTEXT context = {CONTEXT_ALL|CONTEXT_DEBUG_REGISTERS};
    DR7 dr7;

    if (GetThreadContext(hThread, &context))
    {
        dr7 = *(DR7*)&context.Dr7;
        switch (count)
        {
        case 0:
            {
                context.Dr0     = 0;
                dr7.HWBP0_MODE = 0;
                break;
            };
        case 1:
            {
                context.Dr1     = 0;
                dr7.HWBP1_MODE = 0;
                break;
            };
        case 2:
            {
                context.Dr2     = 0;
                dr7.HWBP2_MODE = 0;
                break;
            };
        case 3:
            {
                context.Dr3     = 0;
                dr7.HWBP3_MODE = 0;
                break;
            };
        default:
            return false;
        };
        context.Dr7 = *(PDWORD)&dr7;
        return SetThreadContext(hThread, &context);
    };
    return false;
};
```

لتعامل أسهل مع مسجل Dr7 قمنا بعمل structure لتسهيل الأمر بدلا من إنجاز كل شيء بالـ shifting والـ: bit level

```
struct {
```

```
    ULONG     HWBP0_MODE        :2;
    ULONG     HWBP1_MODE        :2;
    ULONG     HWBP2_MODE        :2;
    ULONG     HWBP3_MODE        :2;
    ULONG     LE                :1;
    ULONG     GE                :1;
    ULONG     __unused          :6;
    ULONG     HWBP0_ACCESS      :2;
    ULONG     HWBP0_LENGTH      :2;
    ULONG     HWBP1_ACCESS      :2;
    ULONG     HWBP1_LENGTH      :2;
    ULONG     HWBP2_ACCESS      :2;
    ULONG     HWBP2_LENGTH      :2;
    ULONG     HWBP3_ACCESS      :2;
    ULONG     HWBP3_LENGTH      :2;
} DR7;
```

3. مثال عملي

سنقوم بعمل مثال loader على crackme بسيط للأخ dj-siba نقتنص منه السريال الصحيح ونتخطى رسالة BadBoy المزعجة. التحدي اسمه "حاول أن تكسرني – تمرين رقم واحد ".

موجود بالمرفقات.

بتحليل سريع للتحدي إليكم المفيد:

```
00403FE9  CALL 00403E60
00403FEE  MOV EDX,dword ptr [ebp - 208]  ---------------------> good serial
00403FF4  POP EAX
00403FF5  CALL 00402F58
00403FFA  JE SHORT 00404011          ---------------------> magic jump
00403FFC  PUSH 0
00403FFE  PUSH 00404050
00404003  PUSH 004040DC
00404008  PUSH 0
0040400A  CALL <JMP.&user32.MessageBoxA> ---------------------> badboy msg
0040400F  JMP SHORT 00404024
00404011  PUSH 0
00404013  PUSH 00404050
00404018  PUSH 00404100
0040401D  PUSH 0
0040401F  CALL <JMP.&user32.MessageBoxA> ---------------------> goodboy msg
00404024  XOR EAX,EAX
```

ما سنفعله هو كالتالي:

- وضع نقطة توقف للـــ execution عند العنوان 0x00403FF4 وبذلك نضمن وجود السريال الصحيح في العنوان الذي يشير إليه المسجل edx.
- وضع نقطة توقف أخرى عند عنوان القفزة التي تؤدي إلى إظهار رسالة عدم التسجيل 0x00403FFA في أول HWBP نقوم بجلب مسجل edx عن طريق GetThreadContext ثم نقوم بقراءة العنوان الذي يشير إليه المسجل من الـــ processعن طريق الدالة ReadProcessMemory ونقوم بإظهار السريال الصحيح.
- في ثاني HWBP نقوم بجلب مسجل الأعلام EFLAGS ونقوم بوضع العلم ZF وذلك كي تتم القفزة المشروطة ونتخطى الرسالة المزعجة.

مسجل EFLAGS أيضاً يمكن التعامل معه عن طريق الـــ bit level لكن قمنا بعمل structure لتسهيل الأمر:

```
struct {
    ULONG    CF              :1;
    ULONG    __unused0       :1;
    ULONG    PF              :1;
    ULONG    __unused1       :1;
    ULONG    AF              :1;
    ULONG    __unused2       :1;
    ULONG    ZF              :1;
    ULONG    SF              :1;
    ULONG    TF              :1;
    ULONG    IF              :1;
    ULONG    DF              :1;
    ULONG    OF              :1;
    ULONG    IOPL            :2;
    ULONG    __unused3       :1;
    ULONG    RF              :1;
    ULONG    VM              :1;
    ULONG    AC              :1;
    ULONG    VIF             :1;
    ULONG    VIP             :1;
    ULONG    ID              :1;
    ULONG    __unused4       :10;
}EFLAGS;
```

ستجد كود اللودر مبرمج بال C في المرفقات (مرفقات الكتاب) وهذه صورة له:



إيجاد السريال بطريقة الهجوم العشوائي

Finding Serial With Brute Force Attack



أرفقنا للاطلاع مجموعة سلاسل المحارف التي يمكن استخدامها في الملف النصي charset.

سنقدم هنا شرحاً مبسطاً لمفهوم الهجوم العشوائي Brute Force من خلال مثال عملي, وسنقوم بتحليل الكود وكتابة كيجن للتمرين.

سيتناول الشرح النقاط الرئيسية التالية :

I. أولاً: مقدمة عن الهجوم العشوائي Introduction To Brute Force Attack

كنا قد ذكرنا سابقاً : أن مصطلح (الهاش) : هو تشفير باتجاه واحد غير قابل للعكس، وكمثال عنه يمكن ذكر: MD5, SHA, Haval, Tiger,...

على سبيل المثال : لو أخذنا الكلمة (AT4RE) وأردنا تشفيرها بواسطة MD5 لنتجت لنا السلسلة المشفرة التالية :

AFA75974BD93160D55DFC4214655E55A

فماذا لو أعطاك صديقك السلسلة المشفرة السابقة, وقال لك: ما هي الكلمة التي نتجت عنها هذه السلسلة بواسطة MD5 ؟ الإجابة الأولى ستكون: لا أعرف !!

ثم قال لك أتحداك أن تعرف ما هي الكلمة التي نتجت عنها هذه السلسلة المشفرة, وحن قلبه عليك قليلاً وقال لك: هي مؤلفة من 5 خانات, وهذه الخانات عبارة عن أرقام وأحرف.

فسألته سريعاً: أهي أحرف كبيرة أم صغيرة ؟ فأجابك ضاحكاً: اعرف ذلك بنفسك!!

بعدها ذهبت حائراً إلى المنزل ومعك هذه الرسالة المشفرة المشؤومة والأفكار تتضارب في رأسك.

جلست أمام الحاسب وبدأت بتجريب بعض الكلمات التي تتعلق بصديقك وشفرتها بـــ MD5 ولكن دون جدوى.

قلت في نفسك: هذا لن ينفع يجب أن أرتب أفكاري فكتبت :

- طول الكلمة : 5 أحرف
- الأحرف المشكلة منها الكلمة هي (حروف صغيرة + كبيرة + أرقام) :

abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

فرحت كثيراً، وقلت في نفسك: إن المسألة عبارة عن احتمالات .

فقط يجب أن أشفر جميع الكلمات المؤلفة من 5 أحرف، ومشكلة من الأحرف السابقة وكلما أشفر كلمة أقارن ناتج التشفير مع الرسالة المشفرة التي أعطاني إياها صديقي حتى أحصل على الكلمة الصحيحة.

قلت في نفسك: الأمر سهل، والأمر المفرح أنك إذا جربت جميع الكلمات فمن المؤكد 100 % أنك ستحصل على الكلمة الصحيحة.

قلت: بسم الله وسأبدأ التجريب، ولكن يجب أن أضع مخططاً للتجريب حتى لا أخطأ في كتابة الكلمات أو أعيد كتابتها، و رسمت الجدول التالي :

Letter 1	Letter 2	Letter 3	Letter 4	Letter 5
a	a	a	a	a
b	b	b	b	b
....	....	....	....	....
A	A	A	A	A
B	B	B	B	B
....	....	....	....	....
1	1	1	1	1
2	2	2	2	2

عندما تنتهي أزيد التي قبلها واحد	عندما تنتهي أزيد التي قبلها واحد	عندما تنتهي أزيد التي قبلها واحد	عندما تنتهي أزيد التي قبلها واحد	عندما تنتهي أكون قد شملت جميع الكلمات

وبدأت بكتابة الكلمات : aaaaa , aaaab, aaaac, ... aaaba, aaabb,aaabc, ...

أظنك تقول: هكذا سأحتاج إلى سنة حتى أحصل عليها وأرى عدد الاحتمالات (عدد الأحرف 62 وعدد حروف الكلمة 5).

إذن : 62^5 = 916,132,832 (رقم كبير!! تحتاج إلى عمر ذو القرنين حتى تحصل على الكلمة).

لا بد من برمجة العملية وترك الحاسب يجرب ذلك عنك ويعطيك النتيجة جاهزة، أو تجد برنامجاً على النت مخصصاً لهذه العمليات.

إذن كلمة الهجوم العشوائي جاءت من تجريبنا تشفير كل الاحتمالات الممكنة حتى نحصل على رسالة مشفرة مساوية للرسالة المشفرة التي بين يدينا.

- هناك برمجيات جاهزة لمثل هذه العمليات نذكر منها : RainbowCrack , Cain
- طبعاً هذه البرمجيات طورت طرائق لتسريع البحث , باعتمادها على بيانات للبحث معدة مسبقاً نذكر منها :
  - World List (Using By Hackers , Name:Pass)
  - Rainbow Tables (For Hash)
- يجدر بنا أن نذكر هنا أن بعض الجداول قد يصل حجمها إلى 4 غيغا فما فوق.
- كما أنه كلما زاد طول الكلمة المشفرة زاد الزمن المطلوب لإيجاد الكلمة نتيجة زيادة عدد الاحتمالات.

الهجوم العشوائي (Brute Force Attack) :
هو طريقة للحصول على الكلمة الأصلية المشفرة من خلال معرفة السلسلة المشفرة وهاش التشفير, بتجريب تشفير جميع الاحتمالات التي يمكن أن تتألف منها الكلمة، ومقارنة الناتج مع السلسلة المشفرة التي بين يدينا.

2. ثانياً: تحليل الكود Code Analysis



اضغط على الصورة للحصول على الهدف الذي طبق عليه الدرس.

– لن ندخل في الفحص وإيجاد التشفير وإيجاد عنوان بداية الإجراء فقد ذكرنا ذلك مراراً.

– لندخل البيانات: الاسم : Roben ، والسريال: 1234 ولنتابع.

- بداية الإجراء :

```
00453F28  PUSH  EBP          بداية الأجراء
00453F29  MOV   EBP, ESP
```

- مقارنات فيما إذا كان طول الاسم أو السريال أقل من أربعة أحرف :

```
00453F53  CALL  00432368
00453F58  CMP   [LOCAL.6], 0                    مقارنة اذا الأسم فارغاً
00453F5C  JE    SHORT 00453F94
00453F5E  LEA   EDX, [LOCAL.7]
00453F61  MOV   EAX, DWORD PTR DS:[EDI+2F8]
00453F67  CALL  00432368
00453F6C  MOV   EAX, [LOCAL.7]
00453F6F  CALL  0040410C
00453F74  CMP   EAX, 4                          مقارنة طول الأسم فيما اذا كان اقل من اربعة حارف
00453F77  JL    SHORT 00453F94
00453F79  LEA   EDX, [LOCAL.8]
00453F7C  MOV   EAX, DWORD PTR DS:[EDI+2FC]
00453F82  CALL  00432368
00453F87  MOV   EAX, [LOCAL.8]
00453F8A  CALL  0040410C
00453F8F  CMP   EAX, 4                          مقارنة طول السريال فيما اذا كان اقل من اربعة حارف
00453F92  JGE   SHORT 00453FA9
00453F94  MOV   EAX, DWORD PTR DS:[EDI+2FC]
00453F9A  MOV   EDX, 4540E8                     ASCII " [ Error ] Incorrect Input ...."
```

- في هذا الاستدعاء يتم أخذ المحرف الأول والثالث من الاسم وثم تجمع وتضاف لها القيمة واحد :

```
00453FC2  CALL  00453D38            أخذ الحرف الأول و الثالث من الأسم و تحويلها الى مكس
00453FC7  LEA   EDX, [LOCAL.10]
```

- إذا دخلنا إلى هذا الاستدعاء (F7) نلاحظ ذلك, مع الانتباه إلى أن المنقح يأخذ بترتيب عكسي :

```
00453D60  MOV     ECX, 4                        أخذ أول أربع حارف من الأسم
00453D65  MOV     EDX, 1
00453D6A  MOV     EAX, [LOCAL.1]
00453D6D  CALL    0040436C
00453D72  MOV     EAX, DWORD PTR DS:[EBX]
00453D74  MOVZX   EAX, BYTE PTR DS:[EAX]        أخذ الحرف الأول
00453D77  MOV     EDX, DWORD PTR DS:[EBX]
00453D79  MOVZX   EDX, BYTE PTR DS:[EDX+2]      أخذ الحرف الثالث
00453D7D  ADD     EAX, EDX                      تجميع الحرفين
00453D7F  INC     EAX
00453D80  LEA     ECX, [LOCAL.2]
00453D83  MOV     EDX, 2
00453D88  CALL    00407D7C                      تحويل الناتج الى مكس
00453D8D  MOV     EDX, [LOCAL.2]
```

--- الناتج الأول : B5

Roben > Robe > Rb > R=52 hex , b = 62 hex , + 1 > B5

- في هذا الاستدعاء يتم أخذ المحرف الثاني والرابع من الاسم وثم تجمع وتضاف لها القيمة واحد :

```
00453FDB  CALL    00453DC0          أخذ الحرف الثاني و الرابع من الأسم و تحويلها الى مكس
00453FE0  LEA     EDX, [LOCAL.3]
```

- إذا دخلنا إلى هذا الاستدعاء (F7) نلاحظ ذلك, مع الانتباه إلى أن المنقح يأخذ بترتيب عكسي :

```
00453DFC  MOVZX   EAX, BYTE PTR DS:[EAX+1]      أخذ الحرف الثاني
00453E00  MOV     EDX, DWORD PTR DS:[EBX]
00453E02  MOVZX   EDX, BYTE PTR DS:[EDX+3]      أخذ الحرف الرابع
00453E06  ADD     EAX, EDX                      تجميع الحارف
00453E08  INC     EAX
00453E09  LEA     ECX, [LOCAL.2]
00453E0C  MOV     EDX, 2
00453E11  CALL    00407D7C                      تحويل الناتج الى مكس
00453E16  MOV     EDX, [LOCAL.2]
```

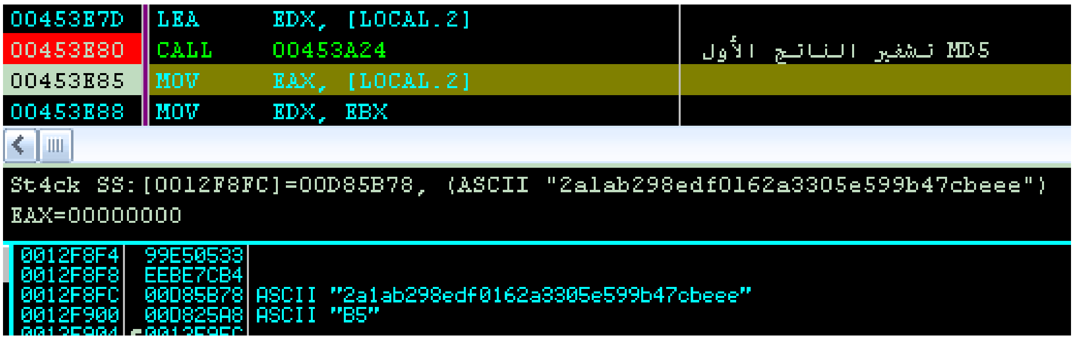
--- الناتج الثاني : D5

Roben > Robe > oe > o=6F hex , e = 65 hex , + 1 > D5

- في هذا الاستدعاء يتم تشفير الناتج الأول بوساطة MD5 وتحويل الناتج إلى أحرف كبيرة :

```
00453FE6  CALL    00453E48          تشفير الناتج الأول
00453FEB  LEA     EAX, [LOCAL.11]
```

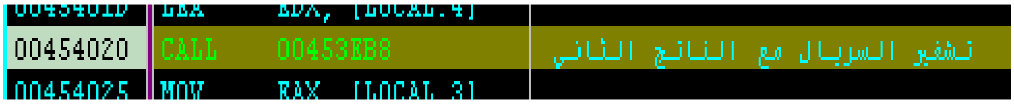
- إذا دخلنا إلى هذا الاستدعاء (F7) نلاحظ ذلك :



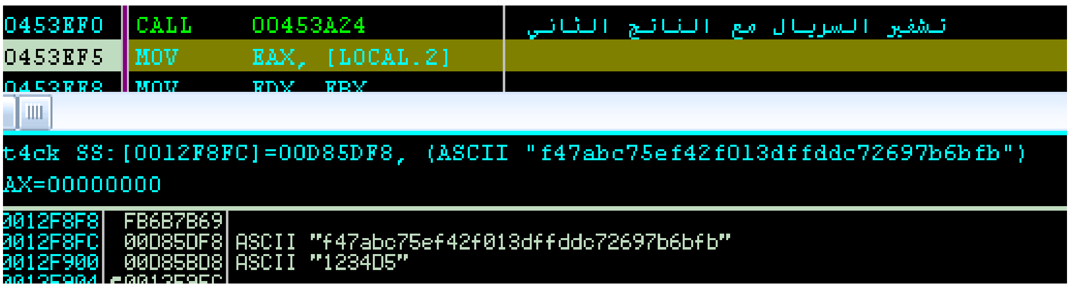
- في هذا الاستدعاء يتم إضافة أول أربعة أحرف من السريال مع الناتج الثاني :



- في هذا الاستدعاء يتم تشفير السريال مع الناتج الثاني 1234D5 :



- إذا دخلنا إلى هذا الاستدعاء (F7) نلاحظ ذلك :



- و أخيراً نجد حلقة لمقارنة كل محرف مع ما يقابله في ناتج التشفيرين، فإذا كانت عدد الأحرف المتشابهة أكبر من 4 ظهرت لنا رسالة النجاح :



3. ثالثاً: تحليل خوارزمية إيجاد السريال

1) أخذ أول أربعة أحرف من الاسم (ما يهمنا هي أول أربعة أحرف من الاسم).

2) أخذ الحرفين الأول والثالث وجمعهما وإضافة واحد لهما، ويكون الناتج بالهكس.

3) تشفير الناتج بـ MD5 (التشفير الأول) وتحويله لأحرف كبيرة .

4) أخذ الحرفين الثاني والرابع وجمعهما وإضافة واحد لهما، ويكون الناتج بالهكس .

5) إضافة الراتج السابق إلى أول أربع أحرف من السريال (إذا السريال مؤلف من 4 أحرف).

6) تشفير الناتج بـ MD5 (التشفير الثاني) وتحويله لأحرف كبيرة .

– يجب أن تتشابه خمس خانات فما فوق من التشفير الأول مع التشفير الثاني.

– علينا تجريب الكلمات المؤلفة من 4 أحرف ثم نضيف لها الناتج الثاني ونشفرها ونقارن فيما إذا كان هناك خمس خانات متشابهة.

– ومن هنا اكتسب التمرين سمة (Brute Force Attack).

4. رابعاً: طرائق التفكير في كتابة الكيجن Brute Forcer

– الطريقة المنهجية: هي الطريقة المذكورة في المقدمة، وهي طريقة عشوائية منظمة، وهي تسمح لك بشمول كافة الاحتمالات ونتائجها مضمونة، لكن من مساوئها :

- تحتاج وقتاً كبيراً نسبياً (وذلك يتعلق بطول السريال وطول سلسلة المحارف).
- تتبع ترتيب منتظم , وبالتالي تبقى الأحرف البعيدة (مثال الكلمة المبتدئة بحرف Z) لوقت طويل.

– الطريقة العشوائية: وهي تعتمد على اختيار ترتيب عشوائي لكل حرف من السريال, فلا تتقيد بالترتيب المتبع، ولكن أحرف الكلمة تظل محصورة ضمن سلسلة المحارف المختارة، ولكن قد تتمتع بسرعة عالية في البحث، وذلك تبعاً للحالة المدروسة، ومن مساوئها :

- عدم شمول كافة الاحتمالات.
- تكرار الكلمات وبالتالي معالجات إضافية بدون مبرر.

– طبعاً يجب أن لا ننسى البرامج المذكورة في المقدمة, فهي مخصصة واحترافية لكسر الهاشات، ولكنها للأسف لا تنفع في تمريننا هذا فهي توجد قيمة مطابقة للهاش ولا يمكنها البحث في الخانات الفرعية.

5. الخاتمة

الآن المعلومات كلها بين يديك ويمكن أن تبرمجها بأي طريقة أحببت وباللغة البرمجية التي تتقنها.

– أرفقنا مع الدرس السورس كود لكيجن التمرين, وهو مكتوب بلغة دلفي7, واعتمدنا فيه على الطريقة العشوائية لسرعتها .

للاطّلاع أرفقنا مجموعة سلاسل المحارف التي يمكن استخدامها في الملف النصي charset.

طرق الـ Anti RCE ووسائل التغلب عليها

I. مقدمة

فك الضغط اليدوي يعتبر فرع من أهم فروع الهندسة العكسية وأكثرها أثارة، ويتطلب خبرة من الكراكر ومعرفة جيدة بالنظام وخباياه. ففي بعض الأحيان يحتاج الكراكر إلى دراسة مفصلة من الداخل لنظام التشغيل لكي يستطيع حل مشكلة مع مضادات الهندسة العكسية Anti reversing tricks قام بوضعها برنامج الضغط أو التشفير في وجه الكراكر ليصعب عملهِ ويصعب عملية فك الضغط. وخلال هذا الجزء من الكتاب سوف نعرض أهم طرق الـــ Anti Unpacking أو الـــ Anti reversing كما سنعرض لكم الطرق المضادة وطرق التغلب عليها سواءً يدوياً أو باستخدام برامج الهندسة العكسية المختلفة.

هناك مصطلحات يصعب ترجمتها إلى اللغة العربية لذلك تم كتابتها كما هي بالإنجليزية.

اضغط الصورة لتحميل المرفقات.

وسيتم تقسيم العمل إلى عدة أجزاء كالتالي :

1) وسائل كشف المنقح DEBUGGER DETECTION .
2) وسائل كشف نقاط التوقف والتعديل على البرنامج BREAKPOINT AND PATCHING DETECTION .
3) وسائل تشويه الكود ومنع تحليله ANTI-ANALYSIS .
4) وسائل أخرى متقدمة ADVANCED AND OTHER TECHNIQUES .

2. وسائل كشف المنقح DEBUGGER DETECTION

هذه الجزئية يتم استخدمها من قبل برنامج الحماية لمعرفة ما إذا كان البروسيس Process الحالية والمراد حمايتها يتم تنقيحها خلال ديبغر أم لا ؟ وتختلف طرق كشف المنقح من الطرق التقليدية إلى طرق متقدمة تستخدم الــ Kernel Mode لكشف الديبغر.

2.1 PEB.BeingDebugged Flag: IsDebuggerPresent[]

2.1.1 الشرح

معظم برامج الحماية تستخدم هذه الطريقة لكشف المنقح، وتتم عن طريق فحص الــ BeingDebugged flag في بنية معروفة باسم Process Environment Block (PEB) ويوجد دالة داخل مكتبة الــ Kernel32 تقوم بعمل هذه المهمة وهي دالة IsDebuggerPresent حيث تقوم الدالة بفحص العنصر BeingDebugged لمعرفة ما إذا كانت الــ Process يتم تنقيحها بواسطة user-mode debugger أم لا؟

2.1.2 مثال

والكود التالي يوضح عملية استخدام دالة IsDebuggerPresent أو الكشف عن العنصر PEB.BeingDebugged flag مباشرة وذلك لكشف المنقح :

```
;call kernel32!IsDebuggerPresent()
call     [IsDebuggerPresent]
test     eax,eax
jnz      .debugger_found

;check PEB.BeingDebugged directly
mov      eax,dword [fs:0x30]       ;EAX = TEB.ProcessEnvironmentBlock
movzx    eax,byte [eax+0x02]     ;AL  = PEB.BeingDebugged
test     eax,eax
jnz      .debugger_found
```

وبما أن هذه الأكواد تعتبر مكشوفة في البرنامج، لذلك يقوم برنامج الحماية بتشفير هذه الجزئية وتشويه الكود حتى لا يمكن التعرف عليه. سنعرف طرق تشفير الكود لاحقا.

2.1.3 وسائل التغلب على هذه الطريقة

يتم التغلب على هذه الطريقة بسهولة بشكل يدوي، وذلك بتعديل قيمة الــ PEB.BeingDebugged flag إلى 0 ولإظهار قيمة الــ PEB.BeingDebugged flag في الــ OllyDbg اذهب إلى جذي الــ Data ثم اضغط Ctrl + G ثم اكتب fs:[30] بالإضافة إلى أن هذا الأمر في الــ OdbgScript وهو dbh يقوم بهذه المهمة.

أيضا يمكن التغلب على هذه الطريقة بواسطة OllyDbg Advanced وهي إضافة معروفة لبرنامج OllyDbg حيث تحتوي على خيار يقوم بوضع 0 في قيمة الــ BeingDebugged field



بالإضافة إلى وجود Plugin منفصلة تقوم بهذه المهمة وهي IsDebuggerPresent Plugin

2.2 PEB.NtGlobalFlag, Heap Flags

2.2.1 الشرح

تحتوي بنية الــ PEB السابقة الذكر على عنصر آخر يسمى NtGlobalFlag عند الأوفست offset 0x68 وهو أيضاً يستخدمه برنامج الحماية لكشف الديبغر. وفي الطبيعي إذا كان لدينا Process تعمل بشكل طبيعي بدون الديبغر تكون قيمة الــ NtGlobalFlag تساوي 0 أما إذا كانت الــ Process تعمل من خلال ديبغر تكون القيمة 70 بالهكس وهذه القيمة ناتجة من عملية OR المنطقية للعناصر الثلاثة :

FLG_HEAP_ENABLE_TAIL_CHECK (0x10)
FLG_HEAP_ENABLE_FREE_CHECK (0x20)
FLG_HEAP_VALIDATE_PARAMETERS (0x40)

أما بالنسبة لــ Heap Flags فإنه بسبب الرايات التي تم تغيرها في الــ NtGlobalFlag فإن هذا يؤثر على العديد من الأعلام في الــ Heap Flags وبصفة عامة الأعلام التي تتغير في الــ heap للبروسيس PEB.ProcessHeap تأخذ القيمة 2 بالهكس والقيمة 0 على التوالي. أما في حالة وجود ديبغر فإن القيم تكون 50000062 بالهكس، ثم 40000060 وبصفة عامة يتم تعديل هذه الأعلام عند إنشاء الــ Heap في حالة وجود ديبغر:

HEAP_TAIL_CHECKING_ENABLED (0x20)

HEAP_FREE_CHECKING_ENABLED (0x40)

2.2.2. مثال

المثال التالي يوضح كيف يمكن استخدام هذه الطريقة وهو يفحص قيمة الـ PEB.NtGlobalFlag لعرف هل تساوي: صفر أم لا ؟ وهل تم تعديل أي أعلام في الـ PEB.ProcessHeap ؟

```
;ebx = PEB
mov     ebx,[fs:0x30]

;Check if PEB.NtGlobalFlag != 0
cmp     dword [ebx+0x68],0
jne     .debugger_found
;eax = PEB.ProcessHeap
mov     eax,[ebx+0x18]

;Check PEB.ProcessHeap.Flags
cmp     dword [eax+0x0c],2
jne     .debugger_found

;Check PEB.ProcessHeap.ForceFlags
cmp     dword [eax+0x10],0
jne     .debugger_found
```

2.2.3. وسائل التغلب على هذه الطريقة

يتم بالتعديل على قيم الـ PEB.NtGlobalFlag والـ PEB.HeapProcess إلى قيمهم الأصلية أي في حال عدم وجود الديبغر، والكود التالي هو لـ OdbgScript وهو يقوم بهذه المهمة :

```
var     peb
var     patch_addr
var     process_heap

//retrieve PEB via a hardcoded TEB address (first thread: 0x7ffde000)
mov     peb,[7ffde000+30]

//patch PEB.NtGlobalFlag
lea     patch_addr,[peb+68]
mov     [patch_addr],0

//patch PEB.ProcessHeap.Flags/ForceFlags
mov     process_heap,[peb+18]
lea     patch_addr,[process_heap+0c]
mov     [patch_addr],2
lea     patch_addr,[process_heap+10]
mov     [patch_addr],0
```

أيضاً إضافة OllyAdvanced بها خيارات للتغلب على هذه الطريقة :



أيضاً إضافة الـ Poison بها خيارات للتغلب على هذه الطريقة :



2.2.4 الشرح

في هذه الطريقة تستخدم دالة CheckRemoteDebuggerPresent التابعة لمكتبة Kernel32 لمعرفة ما إذا كان البروسيس تعمل من خلال ديبغر أم لا؟ وهذه الدالة تقوم داخلياً باستدعاء دالة أخرى وهي NtQueryInformationProcess من مكتبة الـ ntdll وهذه الدالة بها بارامیتر وهو ProcessInformationClass parameter ويتم وضع القيمة ProcessDebugPort (7) فيه. فإذا كانت القيمة في DebugPort لا تساوي 0 فهذا يعني أن البروسيس الحالية تعمل من خلال ديبغر.

وهذا شرح للدالة CheckRemoteDebuggerPresent :

حيث أول بارامیتر هو مقبض البروسيس وثاني بارامیتر من نوع Boolean ويتم فيه وضع القيمة True إذا كان البروسيس يتم تنقيحها.

```
BOOL CheckRemoteDebuggerPresent (
  HANDLE hProcess,
  PBOOL  pbDebuggerPresent
);
```

أما الدالة NtQueryInformationProcess فتحتوي على 5 بارامیتر وهي كالآتي:

```
NTSTATUS NTAPI NtQueryInformationProcess(
```

```
HANDLE          ProcessHandle,
PROCESSINFOCLASS ProcessInformationClass,
PVOID          ProcessInformation,
ULONG           ProcessInformationLength,
PULONG          ReturnLength
);
```

2.2.5. مثال

المثال التالي يوضح استخدام الدالتين السابقتين للكشف عن وجود المنقح :

```
.data
krnl db "kernel32.dll",0h
chkrdbg db "CheckRemoteDebuggerPresent",0h

.data?
IsItPresent dd ?

.code
PUSH offset krnl ;kernel32.dll
CALL LoadLibrary

;CheckRemoteDebuggerPresent
PUSH offset chkrdbg
PUSH EAX
CALL GetProcAddress

;IsItPresent variable will store the resault
PUSH offset IsItPresent
PUSH -1
CALL EAX
MOV EAX,DWORD PTR[IsItPresent]
TEST EAX,EAX
jne DebuggerFound

; using ntdll!NtQueryInformationProcess(ProcessDebugPort)
lea     eax,[.dwReturnLen]

push    eax                    ;ReturnLength
push    4                       ;ProcessInformationLength
lea     eax,[.dwDebugPort]
push    eax                     ;ProcessInformation
push    ProcessDebugPort        ;ProcessInformationClass (7)
push    0xffffffff              ;ProcessHandle
call    [NtQueryInformationProcess]

cmp     dword [.dwDebugPort],0
jne     .debugger_found
```

2.2.6. وسائل التغلب على هذه الطريقة

أول هذه الحلول هو وضع نقطة توقف عند الدالة NtQueryInformationProcess وعند الوقوف على نقطة التوقف يتم تعديل قيمة الـــ ProcessInformation وجعلها 0 وهذا مثال بالـ OdbgScript يوضح هذه العملية :

```
var     bp_NtQueryInformationProcess

// set a breakpoint handler
eob     bp_handler_NtQueryInformationProcess

// set a breakpoint where NtQueryInformationProcess returns
gpa     "NtQueryInformationProcess", "ntdll.dll"
find    $RESULT, #C21400#    //retn 14
mov     bp_NtQueryInformationProcess,$RESULT
bphws   bp_NtQueryInformationProcess,"x"
run

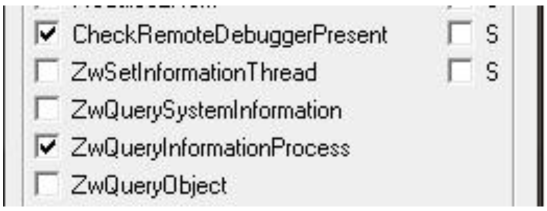
bp_handler_NtQueryInformationProcess:
//ProcessInformationClass == ProcessDebugPort?
cmp     [esp+8], 7
jne     bp_handler_NtQueryInformationProcess_continue

//patch ProcessInformation to 0
mov     patch_addr, [esp+c]
mov     [patch_addr], 0

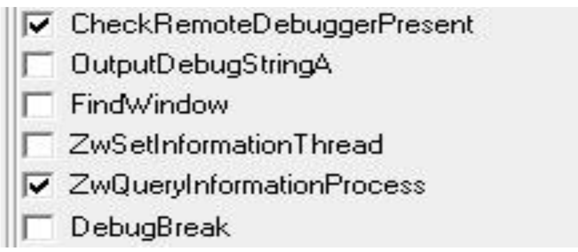
//clear breakpoint
bphwc   bp_NtQueryInformationProcess

bp_handler_NtQueryInformationProcess_continue:
run
```

أما عن الحلول الأوتوماتيكية فإن إضافة الـــ OllyAdvanced تحتوي على خيار للتعامل مع هذه الطريقة :



أيضاً إضافة الـــ Poison تحتوي على نفس الخيارات :



2.3 Debugger Interrupts

2.3.1 الشرح

هذه الطريقة تعتمد على أنه عند تنفيذ تعليمة الــ INT3 Interrupt وهي الخاصة بالـ breakpoint وتعليمة الــ INT1 الخاصة بــ single-step فإنه لن يتم استدعاء الــ exception handler الى بعد تسليم الديبغر للمقبض لل exceptions الناتجة من هذه التعليمات. لذلك يقوم برنامج الحماية بوضع flags داخل الــ exception handler وإذا لم يتم تعديل هذا الــ flag بعد تعليمات الــ INT الخاصة بالـ Interrupt فهذا يعني أن الــ process يتم تنقيحها بواسطة ديبغر.

2.3.2 مثال

هذا المثال يضع القيمة FFFFFFFF بالهكس في المسجل EAX داخل الــ exception handler للتأكد من أن الــ exception handler قد تم استدعاؤه.

```
   ;set exception handler
    push     .exception_handler
    push     dword [fs:0]
    mov      [fs:0], esp

    ;reset flag (EAX) invoke int3
    xor      eax,eax
    int3

    ;restore exception handler
    pop      dword [fs:0]
    add      esp,4

    ;check if the flag had been set
    test     eax,eax
    je       .debugger_found
    :::
.exception_handler:
    ;EAX = ContextRecord
    mov      eax,[esp+0xc]
    ;set flag (ContextRecord.EAX)
    mov      dword [eax+0xb0],0xffffffff
    ;set ContextRecord.EIP
    inc      dword [eax+0xb8]
    xor      eax,eax
    retn
```

2.3.3 وسائل التغلب على هذه الطريقة

يمكن التغلب بسهولة على هذه المشكلة في الــ OllyDbg وذلك بتمرير الــ Exception بالضغط على Shift + F9

أيضاً يوجد حل آخر من داخل إعدادات OllyDbg وذلك باختيار Debugging Options ثم اختيار Exceptions واختيار الخيارات الموجودة في الصورة :



2.4. Timing Checks

2.4.1. الشرح

عندما يتم تنقيح Process معينة فإن الديغر يأخذ العديد من الــ CPU cycles مثل عمليات التتبع بواسطة F8 وبالتتبع فإن الزمن الذي تأخذه الــ Process أثناء التنقيح يكون أكبر من الزمن الذي تأخذه عند تشغيلها بدون منقح؛ لأن عدد الــ CPU cycles يكون أقل. وتستغل برامج الحماية هذه الميزة، وذلك بحساب الزمن بين تنفيذ التعليمات، فإذا أخذت عملية التنفيذ وقتاً أطول من المتوقع (الطبيعي) فهذا يعني أن هذه الــ Process تعمل من خلال ديغر.

2.4.2. مثال

في هذا المثال يقوم البرنامج بعمل حساب للوقت وذلك باستخدام الــ RDTSC أي الــ Read Time-Stamp Counter قبل وبعد تنفيذ العديد من التعليمات وبعد ذلك يحسب قيمة وهي تسمي دلتا delta وهي عبارة عن كمية التعليمات التي تم تنفيذها بين الــ RDTSC الموجودة قبل وبعد الكود، وبالتالي الزمن المستغرق في العملية.

```
rdtsc
mov     ecx,eax
mov     ebx,edx
;... more instructions
nop
push    eax
pop     eax
nop
;... more instructions
;compute delta between RDTSC instructions
rdtsc

;Check high order bits
cmp     edx,ebx
ja      .debugger_found
```

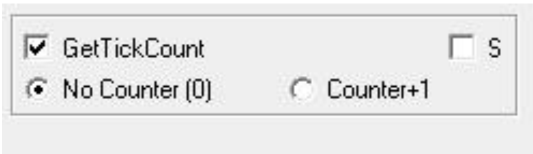
```
;Check low order bits
sub       eax,ecx
cmp       eax,0x200
ja        .debugger_found
```

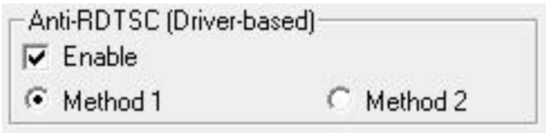
ومن الممكن استخدام الدالة GetTickCount التابعة لمكتبة الـــ kernel32 لإنجاز هذه المهمة.

2.4.3 وسائل التغلب على هذه الطريقة

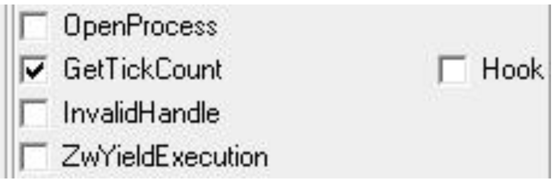
أول هذه الحلول هو تحديد مكان الـــ timing checks ومحاولة تجنب المرور بكود التحقق أو وضع نقطة توقف على الدالة GetTickCount ومحاولة تغيير ناتج الدالة.

أيضاً إضافة الـــ OllyAdvanced التي تحتوي على حل لهذه المشكلة، وذلك عن طريق استخدام درايفر kernel mode driver ولكن في بعض الأحيان يؤدي هذا الدرايفر إلى حدوث مشاكل؛ مثل الـــ BSOD وهي شاشة الموت الزرقاء.





أيضا إضافة الـــ Poison بها، هي الأخرى خيار للتعامل مع هذه الطريقة :



2.5 SeDebugPrivilege

2.5.1 الشرح

بصفة عامة فإن أي Process تعمل بشكل طبيعي في النظام يكون لديها ميزة الـــ SeDebugPrivilege غير مفعلة. أما عند تشغيل الـــ Process من خلال أي User Mode Debugger مثل الـــ OllyDbg أو الـــ WinDbg فإن ميزة الـــ SeDebugPrivilege تكون مفعلة. ولذلك تستغل برامج الحماية هذه الطريقة، وذلك بالكشف عن الـــ SeDebugPrivilege

هل هي مفعلة أم لا؟ وذلك عن طريق محاولة فتح الــ Process المسماة CSRSS.EXE فإذا تمكن من فتح هذه الــ CSRSS.EXE فهذا يعني أن الــ Process المراد حمايتها لديها الــ SeDebugPrivilege مفعلة، مما يعني أنه يتم تنقيحها من خلال ديبغر.

والسبب في ذلك أن الــ Process المسماة CSRSS.EXE تسمح للنظام بفتحها فقط إذا كانت process لديها الــ SeDebugPrivilege .

2.5.2. مثال

هذا مثال يوضح الطريقة السابقة، وهو يقوم باستخدام الدالة CsrGetProcessId من مكتبة ntdll وذلك للحصول على الــ Process ID للـ CSRSS.EXE ولكن ملؤن أن يقوم برنامج الحماية بتحديد الــ ID دون استخدام هذه الدالة، بل يقوم بالمهمة بشكل يدوي، ثم يستخدم الدالة OpenProcess فإذا نجحت الدالة فهذا يعني أن الــ SeDebugPrivilege مفعلة مما يعني أن الــ Process يتم تنقيحها.

```
;query for the PID of CSRSS.EXE
call     [CsrGetProcessId]

;try to open the CSRSS.EXE process
push     eax
push     FALSE
push     PROCESS_QUERY_INFORMATION
call     [OpenProcess]

;if OpenProcess() was successful,
;process is probably being debugged
test      eax,eax
jnz      .debugger_found
```

2.5.3. وسائل التغلب على هذه الطريقة

هناك حل يدوي وهو وضع نقطة توقف عرج الدالة NtOpenProcess وتغيير ناتج الدالة، أي الــ EAX إلى القيمة C0000022 ه كس، وهي تعني STATUS_ACCESS_DENIED أي أن عملية فتح الــ Process فشلت.

2.6. Parent Process

2.6.1. الشرح

بشكل عام يكون الــ Parent لأي Process تعمل بشكل طبيعي في النظام هو الــ explorer.exe لذلك يقوم برنامج الحماية باستغلال هذه النقطة ومعرفة الــ Parent للبروسيس المراد حمايتها، فإذا كان أي شيء آخر غير الــ explorer.exe فهذا يعني أن البروسيس ربما تكون تعمل من خلال الديبغر.

وهذه الطريقة يتم عملها كالتالي :

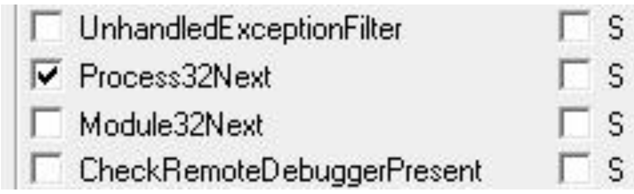
- أولاً يقوم برنامج الحماية بجلب الــ PID للبروسيس الحالية، وذلك عن طريق البنية TEB من خلال العنصر TEB.ClientId أو مباشرة من خلال الدالة GetCurrentProcessId .
- ثم يقوم بعرض جميع الــ Processes التي تعمل في النظام عن طريق الدالتين Process32First/Next ثم يأخذ الــ PID لـ explorer.exe عن طريق PROCESSENTRY32.szExeFile ثم نفس الشيء مع الــ parent process للبروسيس التي يريد حمايتها بواسطة PROCESSENTRY32.th32ParentProcessID.
- ثم يقارن: هل الــ PID لـ parent process لا تساوي الــ PID لـ explorer.exe ؟ فإذا كانت لا تساويها فهذا يعني أن البروسيس تعمل من خلال الدييغر.

2.6.2. وسائل التغلب على هذه الطريقة

تحتوي إضافة الــ OllyAdvanced على خيار للتعامل مع هذه الطريقة، وهو يعتمد على إفشال الــ Process32Next مما لا يمكن برنامج الحماية من معرفة الــ PID :

```
7C863B5F   33C0      XOR EAX,EAX     kernel32!Process32NextW
7C863B61   C2 0800   RETN 8
7C863B64   83EC 0C   SUB ESP,0C
```

وذلك بتصفير الــ EAX ثم وضع RETN عند مكان هذه الدالة.



أيضا إضافة الــ Poison تحتوي على خيار للتعامل مع هذه الطريقة :



2.7 DebugObject: NtQueryObject

2.7.1 الشرح

بدلاً من معرفة ما إذا كانت الــ Process يتم تنقيحها بواسطة ديبغر أم لا؟ فإن هذه الطريقة تعتمد على معرفة ما إذا كان هناك ديبغر يعمل أم لا بغض النظر عن الــ Process ؟

ويتم ذلك عن طريق فحص عدد الــ kernel objects من النوع DebugObject لأنه عندما يتم تنقيح أي برنامج يتم إنشاء object في الكرنل kernel تسمى DebugObject .

ويتم معرفة عدد الــ DebugObject عن طريق الدالة NtQueryObject من مكتبة ntdll وهذه الدالة تستقبل 5 باراميتر لكي نقوم بمعرفة كل الــ objects types يتم وضع الــ ObjectHandle بــ NULL والــ ObjectInformationClass بــ ObjectAllTypeInformation .

NTSTATUS NTAPI NtQueryObject(
HANDLE ObjectHandle,
OBJECT_INFORMATION_CLASS ObjectInformationClass,
PVOID ObjectInformation,
ULONG Length,
PULONG ResultLength
);

وهذه الدالة تقوم بإرجاع البنية المعروفة باسم OBJECT_ALL_INFORMATION وهذه البنية تحتوي على عنصر الــ NumberOfObjectsTypes

```
typedef struct _OBJECT_ALL_INFORMATION {
ULONG NumberOfObjectsTypes;
OBJECT_TYPE_INFORMATION ObjectTypeInformation[1];
}
```

أيضا الــ ObjectTypeInformation عبارة عن مصفوفة لها البنية التالية :

```
typedef struct _OBJECT_TYPE_INFORMATION {
/*[00]*/ UNICODE_STRING TypeName;
/*[08]*/ ULONG TotalNumberOfHandles;
/*[0C]*/ ULONG TotalNumberOfObjects;
//... more fields ...
}
```

بعد ذلك يتم مقارنة العنصر TypeName مع الــ UNICODE string الذي نبحث عنة وهو "DebugObject" وبعد ذلك يتم مقارنة العنصر TotalNumberOfObjects أو TotalNumberOfHandles مع قيمة لا تساوي الصفر.

2.7.2. وسائل التغلب على هذه الطريقة

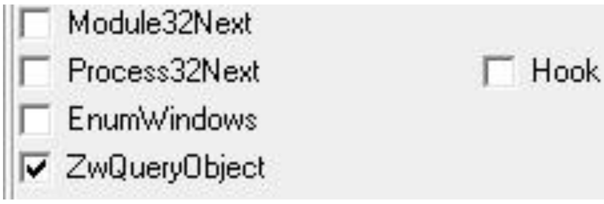
نفس طريقة حل الـــ NtQueryInformationProcess حيث نقوم بوضع نقطة توقف عند الـــ NtQueryObject بعد ذلك يمكن تغيير قيمة الـــ OBJECT_ALL_INFORMATION وخصوصاً العنصر NumberOfbjectsTypes حيث نقوم بتعديله إلى القيمة 0

أيضا يمكن عمل OdbgScript للقيام بهذه المهمة.

وأيضاً إضافة الـــ OllyAdvanced تحتوي على خيار للتعامل مع دالة الـــ NtQueryObject :



وكذلك إضافة الـــ Poison تحتوي على نفس الخيار :



2.8 Debugger Window

2.8.1. الشرح

هي وسيلة أخرى من وسائل كشف المنقح تعتمد على البحث عن اسم نافذة الديبغر. حيث أن أي ديبغر يتم تشغيله تكون النافذة الخاصة به لها عنوان محدد مثل (OLLYDBG for OllyDbg, WinDbgFrameClass for WinDbg) ويتم التعرف على عنوان النافذة بسهولة عن طريق الدالة FindWindow الموجودة في مكتبة User32 أو الدالة FindWindowEx الموجودة في نفس المكتبة.

2.8.2 مثال

هذا مثال بسيط يوضح استخدام الدالة FindWindow لمعرفة ما إذا كان الـــ OllyDbg أو الـــ WinDbg يعمل في النظام أم لا؟

```
.data?
.szWindowClassOllyDbg    db "OLLYDBG",0
.szWindowClassWinDbg     db "WinDbgFrameClass",0
.code
push     NULL
push     .szWindowClassOllyDbg
call     [FindWindowA]
test     eax,eax
jnz      .debugger_found
push     NULL
push     .szWindowClassWinDbg
call     [FindWindowA]
test     eax,eax
jnz      .debugger_found
```

2.8.3 وسائل التغلب على هذه الطريقة

الطريقة اليدوية: عن طريق وضع نقطة توقف عند الدالة FindWindow وFindWindowEx ثم تغيير قيمة الـــ lpClassName string parameter وهي التي تحوي اسم الديبغر أو وضع قيمة 0 في الـــ EAX حتى لا تتم القفزة.

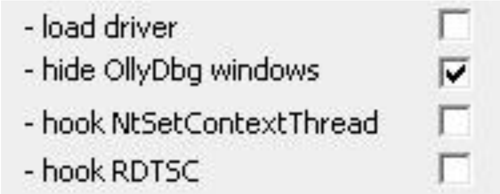
أو عن طريق إضافة الـــ OllyAdvanced :



أو إضافة الـــ poison :



وإضافة الـــ Phantom بها خيار لإخفاء نافذة OllyDbg :



2.9. Debugger Process

2.9.1. الشرح

هي طريقة أخرى لمعرفة ما إذا كان هناك ديبغر يعمل في النظام أم لا؟ وهي طريقة مشابهة للطريقة السابقة، حيث يقوم برنامج الحماية بعمل جرد لكل الـــ Processes التي تعمل في النظام ويقارنهم مع أسماء الـــ Processes الخاصة بالمنقحات؛ مثل: (e.g. OLLYDBG.EXE, windbg.exe, etc.) ويتم ذلك عن طريق الدالة Process32First/Next وفحص image name إذا كان لديبغر أم لا؟

في بعض برامج الحماية يتم استخدام طريقة أصعب قليلاً، وهي عن طريق استخدام الدالة ReadProcessMemory ثم البحث عن سترينغ معين يكون معروف داخل الديبغر؛ مثل e.g. "OLLYDBG" لأنه لو قام الديبغر بتغيير اسم ملف الـــ exe الخاص بالديبغر فستفشل الطريقة الأولى لذلك يتم استخدام الطريقة الثانية

2.9.2. وسائل التغلب عليهذه الطريقة

نفس طريقة حل الـــ parent process check وهي مهاجمة الدالة Process32NextW لمنع برنامج الحماية من جرد الـــ Processes .

2.10. Device Drivers

2.10.1. الشرح

هذه الطريقة هي طريقة تقليدية لمعرفة ما إذا كان هناك Kernel Mode Debugger يعمل في النظام أم لا؟ وأمثلة الـــ Kernel Mode Debugger برنامج الـــ SoftICE وهذه الطريقة غاية في البساطة؛ حيث تعتمد الفكرة على محاولة إرجاع مقبض لملف الدرايفر الخاص بالديبغر وذلك باستخدام الدالة CreateFile

2.10.2. مثال

```
push      NULL
```

```
push     0
push     OPEN_EXISTING
push     NULL
push     FILE_SHARE_READ
push     GENERIC_READ
push     .szDeviceNameNtice
call      [CreateFileA]
cmp      eax,INVALID_HANDLE_VALUE
jne      .debugger_found

.szDeviceNameNtice  db "\\.\NTICE",0
```

أيضاً بعض برامج الحماية الحديثة تستخدم طريقة الـ Device driver detection لكشف برامج مراقبة النظام أمثال الــ Regmon and Filemon .

2.10.3 وسائل التغلب على هذه الطريقة

وضع نقطة توقف عند الدالة kernel32!CreateFileFileW ومحاولة تغيير الــ FileName parameter أو تغيير رجع الدالة إلى INVALID_HANDLE_VALUE حتى يعتقد برنامج الحماية أن ملف الدرايفر غير موجود.

2.11 OllyDbg: Guard Pages

2.11.1 الشرح

هذه الطريقة خاصة فقط بالـ OllyDbg لأنها تستغل طريقة OllyDbg في acess/write للـ Breakpoint memory

لأن هذا النوع من الــ breakpoint يستخدم الــ guard pages وبواسطة الــ guard pages تمكن برنامج معين من معرفة متى يتم عمل access للذاكرة. ويتم وضع الــ Guard pages بواسطة الــ PAGE_GUARD page protection modifier فإذا كان العنوان المراد الدخول إليه جزءاً من page guard فسيتم رفع الــ STATUS_GUARD_PAGE_VIOLATION (0x80000001) .

وتستغل برامج الحماية أنه عندما يكون هناك Process تعمل من خلال الــ OllyDbg ثم إذا تم محاوله الدخول إلى guard page في الذاكرة فلن يتم تمرير أي exception أيضا، هذا الدخول access سيعامل على أنه memory breakpoint .

2.11.2. مثال

في هذا المثال يقوم الكود بتحديد جزء من الذاكرة، ثم يقوم بكتابة كود إلى هذا الجزء في الذاكرة، ثم تفعيل خاصية الـ PAGE_GUARD ثم يقوم بتشغيل هذا الكود الموجود في هذا الجزء من الذاكرة.

```
; set up exception handler
push     .exception_handler
push     dword [fs:0]
mov      [fs:0], esp

; allocate memory
push     PAGE_READWRITE
push     MEM_COMMIT
push     0x1000
push     NULL
call     [VirtualAlloc]
test     eax,eax
jz       .failed
mov      [.pAllocatedMem],eax

; store a RETN on the allocated memory
mov      byte [eax],0xC3

; then set the PAGE_GUARD attribute of the allocated memory
lea      eax,[.dwOldProtect]
push     eax
push     PAGE_EXECUTE_READ | PAGE_GUARD
push     0x1000
push     dword [.pAllocatedMem]
call     [VirtualProtect]

; set marker (EAX) as 0
xor      eax,eax
; trigger a STATUS_GUARD_PAGE_VIOLATION exception
call     [.pAllocatedMem]
; check if marker had not been changed (exception handler not called)
test     eax,eax
je       .debugger_found
:::

.exception_handler
;EAX = CONTEXT record
mov      eax,[esp+0xc]
;set marker (CONTEXT.EAX) to 0xffffffff
; to signal that the exception handler was called
mov      dword [eax+0xb0],0xffffffff
xor      eax,eax
retn
```

3. وسائل كشف نقاط التوقف والتعديل على البرامج Breakpoint & Patching Detection

3.1 Software Breakpoint Detection

3.1.1 الشرح

نقاط التوقف من هذا النوع هي عبارة عن تغيير في كود البرنامج الهدف عند العنوان المحدد للتوقف عليه. ويتم تغيير كود البرنامج بهذه التعليمة INT3 / Breakpoint Interrupt أي البايت CC بالهكس. وبالطبع تقوم برامج الحماية بالبحث داخل الجزء المحمي من الكود أو داخل كود دوال الـ API عن هذه التعليمة.

3.1.2 مثال

هذا مثال بسيط يوضح فحص الكود للبحث عن نقاط التوقف :

```
cld
mov    edi,Protected_Code_Start
mov    ecx,Protected_Code_End - Protected_Code_Start
mov    al,0xcc
repne  scasb
jz     .breakpoint_found
```

تقوم بعض برامج الحماية بمحاولة خداع الكراكر، وذلك باستبدال سطر المقارنة السابق بهذا السطر :

```
if(byte XOR 0x55 == 0x99) then breakpoint found

// Where: 0x99 == 0xCC XOR 0x55
```

3.1.3 وسائل التغلب على هذه الطريقة

إما محاولة الوصول إلى كود المقارنة وتعديله وإما استخدام الـ hardware breakpoints .

3.2 Hardware Breakpoint Detection

3.2.1 الشرح

نوع آخر من أنواع نقاط التوقف هو الـ hardware breakpoint . ويتم وضع الـ hardware breakpoint بواسطة مسجلات الـ debug registers وهذه المسجلات هي من Dr0 إلى Dr7 . من Dr0 إلى Dr3 تحتوي هذه المسجلات على عناوين لـ 4 نقاط توقف، لذلك تلاحظ في OllyDbg أن آخر عدد للـ hardware breakpoint هو 4 فقط. أما الـ Dr6

فهو علم يبين أي من نقاط التوقف الأربعة تعمل الآن؟ أما الـ Dr7 فهو أيضا علم، ولكن يتحكم في الـ enabling/disabling لنقاط التوقف وأيضاً أي من النقاط سيتوقف عند الـ read/write ؟

وعملية كشف نقاط التوقف hardware breakpoints تحتاج إلى كود كبير جزئياً، وذلك لأن مسجلات الـ debug registers لا يمكن الدخول إليها من الـ Ring 3 أي الـ User Mode لذلك تقوم برامج الحماية بتحليل بنية الـ CONTEXT structure وهي التي تحتوي على قيم الـ debug registers والـ CONTEXT structure يتم قراءتها من البارامتر المسمى ContextRecord الممرر إلى الـ exception handler .

3.2.2 مثال

هذا مثال يوضح طريقة تحليل الـ debug registers :

```
; set up exception handler
push    .exception_handler
push    dword [fs:0]
mov     [fs:0], esp

;eax will be 0xffffffff if hardware breakpoints are identified
xor     eax,eax

; throw an exception
mov     dword [eax],0

; restore exception handler
pop     dword [fs:0]
add     esp,4

; test if EAX was updated (breakpoint identified)
test    eax,eax
jnz     .breakpoint_found

:::

.exception_handler
;EAX = CONTEXT record
mov     eax,[esp+0xc]

;check if Debug Registers Context.Dr0-Dr3 is not zero
cmp     dword [eax+0x04],0
jne     .hardware_bp_found
cmp     dword [eax+0x08],0
jne     .hardware_bp_found
cmp     dword [eax+0x0c],0
jne     .hardware_bp_found
cmp     dword [eax+0x10],0
jne     .hardware_bp_found
jmp     .exception_ret

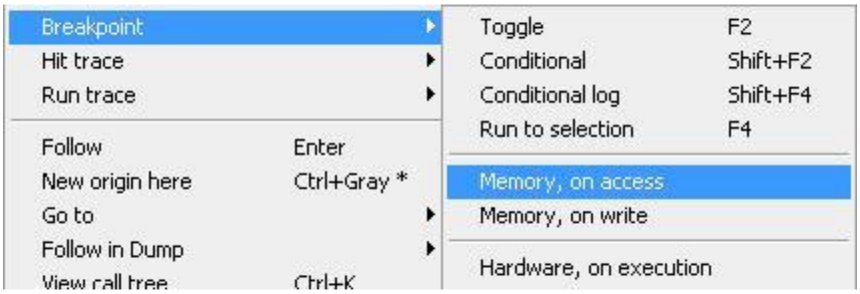
.hardware_bp_found
;set Context.EAX to signal breakpoint found
mov     dword [eax+0xb0],0xffffffff
```

```
.exception_ret
;set Context.EIP upon return
add      dword [eax+0xb8],6
xor      eax,eax
retn
```

بعض برامج الحماية تستخدم الـ debug registers كجزء من مفاتيح فك التشفير decryption keys حيث يقوم برنامج الحماية بمراقبة هذه المسجلات مع اعتبار أنها من المفترض أن تحتوي على القيم 0 فإذا وجد تغيير في قيم هذه المسجلات فإن عملية فك التشفير سوف تفشل، مما يؤدي إلى إنتاج تعليمات خاطئة في كود البرنامج، مما يؤدي إلى عدم عمل البرنامج.

3.2.3. وسائل التغلب على هذه الطريقة

أول طريقة هي استخدام الـ software breakpoints إذا كان ليس هناك فحص من برنامج الحماية عنها. أو محاولة استخدام الـ access/write memory breakpoint نقاط توقف الذاكرة وهي موجودة في الـ OllyDbg



أو يمكن وضع software breakpoint على دوال الـ native APIs إذا كان الأمر يتطلب وضع نقاط توقف على API معينة.

3.3 Patching Detection via Code Checksum Calculation

3.3.1. الشرح

تم شرح هذه الجزئية بالتفصيل في الإصدار الأول من الكتاب في جزء الـ CRC32 لذا سنمر عليها سريعاً.

هذه الطريقة تقوم بالتحقق من كود البرنامج المحمي لكشف أي تعديل على الكود الأصلي. فمثلاً برامج الحمايق تقوم بهذا الفحص للتأكد من أنه لم يتم التلاعب بالكود؛ لأن أي عملية تعديل في الكود الأصلي للبرنامج المحمي تعني أنه قد تم إيقاف أي خاصية من خواص الـ anti-debugging الموجودة في برنامج الحماية.

ليست وظيفة هذه الطريقة كشف التعديل في الكود فقط، بل يمكن استخدمها لكشف الــ software breakpoints لأننا كما تكلمنا من قبل فإن نقاط التوقف ما هي إلا تعديل في كود البرنامج.

3.3.2. مثال

```
mov     esi,Protected_Code_Start
mov     ecx,Protected_Code_End - Protected_Code_Start
xor     eax,eax

.checksum_loop
movzx   ebx,byte [esi]
add     eax,ebx
rol     eax,1
inc     esi
loop    .checksum_loop

    cmp     eax,dword [.dwCorrectChecksum]
    jne     .patch_found
```

3.3.3. وسائل التغلب على هذه الطريقة

إذا كان استخدام الــ code checksum من أجل كشف نقاط التوقف فيمكنك استخدام الــ hardware breakpoints أما إذا كان استخدام الــ code checksum من أجل كشف أي تعديل في كود البرنامج فالحل هو معرفة عنوان الــ checksum routine وتعديلها، ويتم معرفة عنوان الــ checksum عن طريق برامج فحص التشفير أمثال إضافة الــ Kanal الموجودة في الــ PEiD أو برنامج HCD .

4. وسائل تشويه الكود ومنع تحليله ANTI-ANALYSIS

الهدف من هذه الطرق إبطاء عمل الكراكر عن طريق جعل كود البرنامج المحمي غير مفهوم ومشوه. وهذه الطرق تتطلب من اللكواكر الصبر والمهارة للتغلب عليها.

4.1. Encryption and Compression

4.1.1. الشرح

هذه هي الخطوة الأولى والأساسية في حماية الكود، وهدفها منع الكراكر من تحميل البرنامج المحمي في أي disassembler إضافة إلى الصعوبة في تحليل البرنامج، فهي تعتبر محطة الدفاع الأولى.

وتقوم برامج الحماية عادة بتشفير كود برنامج الحماية نفسه وكود البرنامج المراد حمايته. أيضاً طريقة التشفير encryption algorithm تختلف اختلافاً كبيراً باختلاف برامج الحماية؛ فبعضها يكون عبارة عن كود صغير لحلقة XOR وبعضها يكون الكثير من الحلقات المعقدة التي تقوم بالكثير من العمليات الحسابية. أيضاً من فوائد تشفير الكود تضليل برامج الفحص مثل الـ PEiD والـ FastScanner .

9.1.2. مثال

أما عن إجرائيات فك التشفير فتكون عبارة عن حلقات تقوم بعمليات التقاط fetch وحساب وتخزين للتعليمة بعد فك تشفيرها، والمثال التالي يوضح إجرائية فك تشفير بسيطة تستخدم العديد من عمليات الـ XOR :

```
0040A07C  LODS DWORD PTR DS:[ESI]
0040A07D  XOR EAX,EBX
0040A07F  SUB EAX,12338CC3
0040A084  ROL EAX,10
0040A087  XOR EAX,799F82D0
0040A08C  STOS DWORD PTR ES:[EDI]
0040A08D  INC EBX
0040A08E  LOOPD SHORT 0040A07C   ;decryption loop
```

وهذا مثال آخر أكثر تعقيداً :

```
00476056  MOV BH,BYTE PTR DS:[EAX]
00476058  INC ESI
00476059  ADD BH,0BD
0047605C  XOR BH,CL
0047605E  INC ESI
0047605F  DEC EDX
00476060  MOV BYTE PTR DS:[EAX],BH
00476062  CLC
00476063  SHL EDI,CL
::: More garbage code
00476079  INC EDX
0047607A  DEC EDX
0047607B  DEC EAX
0047607C  JMP SHORT 0047607E
0047607E  DEC ECX
0047607F  JNZ 00476056   ;decryption loop
```

الأسطر الظاهرة هي التعليمات المسؤولة عن فك التشفير، أما باقي الأسطر فهي لتضليل الكراكر فقط.

أما عملية الضغط فهدفها الأساسي تقليل حجم البرنامج عن طريق تقليل حجم الكود والبيانات Data ومن أمثلة خوارزميات الضغط التي تستخدم من قبل برامج الحماية خوارزميات الـ NRV (Not Really Vanished) والـ LZMA وهي اختصار لـ (Lempel-Ziv-Markov chain-Algorithm) وتستخدم في برنامج الـ UPX وخوارزمية الـ aPLib وتستخدم في برنامج

الــ FSG وخوارزمية الــ LZMA وتستخدم في برنامج الــ Upack وخوارزمية الــ LZO وتستخدم في برنامج yoda's Protector .

4.1.3. وسائل التغلب على هذه الطريقة

عملية تخطي حلقات فك التشفير والضغط سهلة ويكفي معرفة مكان الحلقة ثم وضع نقطة توقف خارج هذه الحلقة حتى يتوقف البرنامج عندها بعد انتهاء الحلقة.

4.2. Garbage Code and Code Permutation

4.2.1. الشرح

الحقيقة لم نجد ترجمة مناسبة لكلمة Garbage إلا كلمة نفايات، أي ملء الكود الأصلي المراد حمايته بكود آخر غير مفهوم ولا معنى له؛ لأجل تضليل الكراكر. ويتم وضع الــ garbage code في إجرائيات فك الضغط أو إجرائيات كشف المنقح بهدف حماية كود الكشف وإخفائه حتى لا يستطيع الكراكر الوصول إليه بسهولة. وكلما كان الــ garbage code يعطي إيحاء بأن له معنى أو مفيد كلما كان أكثر تأثيراً وفاعلية.

4.2.2. مثال

المثال التالي يوضح عملية فك تشفير لكود تم وضع العديد من الــ garbage code بين الأسطر الفعالة لهذا الإجراء :

```
0044A21A   JMP SHORT sample.0044A21F
0044A21C   XOR DWORD PTR SS:[EBP],6E4858D
0044A223   INT 23
0044A225   MOV ESI,DWORD PTR SS:[ESP]
0044A228   MOV EBX,2C322FF0
0044A22D   LEA EAX,DWORD PTR SS:[EBP+6EE5B321]
0044A233   LEA ECX,DWORD PTR DS:[ESI+543D583E]
0044A239   ADD EBP,742C0F15
0044A23F   ADD DWORD PTR DS:[ESI],3CB3AA25
0044A245   XOR EDI,7DAC77F3
0044A24B   CMP EAX,ECX
0044A24D   MOV EAX,5ACAC514
0044A252   JMP SHORT sample.0044A257
0044A254   XOR DWORD PTR SS:[EBP],AAE47425
0044A25B   PUSH ES
0044A25C   ADD EBP,5BAC5C22
0044A262   ADC ECX,3D71198C
0044A268   SUB ESI,-4
0044A26B   ADC ECX,3795A210
0044A271   DEC EDI
0044A272   MOV EAX,2F57113F
0044A277   PUSH ECX
0044A278   POP ECX
```

```
0044A279  LEA EAX,DWORD PTR SS:[EBP+3402713D]
0044A27F  DEC EDI
0044A280  XOR DWORD PTR DS:[ESI],33B568E3
0044A286  LEA EBX,DWORD PTR DS:[EDI+57DEFEE2]
0044A28C  DEC EDI
0044A28D  SUB EBX,7ECDAE21
0044A293  MOV EDI,185C5C6C
0044A298  MOV EAX,4713E635
0044A29D  MOV EAX,4
0044A2A2  ADD ESI,EAX
0044A2A4  MOV ECX,1010272F
0044A2A9  MOV ECX,7A49B614
0044A2AE  CMP EAX,ECX
0044A2B0  NOT DWORD PTR DS:[ESI]
```

الكود الهام والمسؤول عن عملية فك التشفير في الكود السابق هو :

```
0044A225  MOV ESI,DWORD PTR SS:[ESP]
0044A23F  ADD DWORD PTR DS:[ESI],3CB3AA25
0044A268  SUB ESI,-4
0044A280  XOR DWORD PTR DS:[ESI],33B568E3
0044A29D  MOV EAX,4
0044A2A2  ADD ESI,EAX
0044A2B0  NOT DWORD PTR DS:[ESI]
```

أما العملية الثانية وهي الـــ Code Permutation فتعني استبدال التعليمات البسيطة بأخرى أكثر تعقيداً. وهذا التكتيك مستخدم من قبل برامج الحماية الحديثة. وهذا يتطلب من برنامج الحماية قراءة التعليمة الأصلية ثم توليد عدد من التعليمات المعقدة التي تقوم بنفس الوظيفة.

وهذا مثال بسيط :

```
mov    eax,ebx
test   eax,eax
```

ويتم تحويله إلى تعليمات أخرى تقوم بنفس المهمة :

```
push   ebx
pop    eax
or     eax,eax
```

وفي النهاية عمل توليفة من الــ garbage code والـ Code Permutation لحماية كود معين سيكون له دور كبير في إبطاء عمل الكراكر. وكمثال على ذلك شاهد الكود التالي ما هو إلا عملية كشف منقح بسيطة، لكن استخدمنا فيها تقنية الــ garbage code مع الــ Code Permutation :

```
004018A3  MOV EBX,A104B3FA
004018A8  MOV ECX,A104B412
004018AD  PUSH 004018C1
004018B2  RETN
004018B3  SHR EDX,5
004018B6  ADD ESI,EDX
004018B8  JMP SHORT 004018BA
004018BA  XOR EDX,EDX
004018BC  MOV EAX,DWORD PTR DS:[ESI]
004018BE  STC
004018BF  JB SHORT 004018DE
004018C1  SUB ECX,EBX
004018C3  MOV EDX,9A01AB1F
004018C8  MOV ESI,DWORD PTR FS:[ECX]
004018CB  LEA ECX,DWORD PTR DS:[EDX+FFFF7FF7]
004018D1  MOV EDX,600
004018D6  TEST ECX,2B73
004018DC  JMP SHORT 004018B3
004018DE  MOV ESI,EAX
004018E0  MOV EAX,A35ABDE4
004018E5  MOV ECX,FAD1203A
004018EA  MOV EBX,51AD5EF2
004018EF  DIV EBX
004018F1  ADD BX,44A5
004018F6  ADD ESI,EAX
004018F8  MOVZX EDI,BYTE PTR DS:[ESI]
004018FB  OR EDI,EDI
004018FD  JNZ SHORT 00401906
```

وهذا المثال السابق ما هو إلا صيغة معقدة من هذا الإجراء :

```
00401081  MOV EAX,DWORD PTR FS:[18]
00401087  MOV EAX,DWORD PTR DS:[EAX+30]
0040108A  MOVZX EAX,BYTE PTR DS:[EAX+2]
0040108E  TEST EAX,EAX
00401090  JNZ SHORT 00401099
```

4.2.3. وسائل التغلب على هذه الطريقة

هذه التقنية ما هي إلا وسيلة لإصابة الكراكر بالملل وإضاعة وقته وجهده.

ولتقليل الوقت الضائع استخدم دوال الــ API التي يكثر استدعاؤها من برامج الحماية مثل (VirtualAlloc/VirtualProtect/LoadLibrary/GetProcAddress) ووضع نقاط توقف عليها.

أيضاً استخدم نقاط التوقف من نوع access/write breakpoints لتجنب أي فحص من برنامج الحماية.

4.3. Anti-Disassembly

4.3.1. الشرح

هي طريقة أخرى لتصعيب مهمة الكراكر من خلال جعل برامج التنقيح تفشل في تحليل كود البرنامج، وبالتالي يظهر كود غريب غير مفهوم. وهي طريقة فعالة جداً خاصة إذا ما تم إضافة بعض الـ garbage code والـ permutation .

ومن أمثلة تقنية الـ Anti-Disassembly إضافة بايت garbage وبعد ذلك إضافة قفزة مشروطة conditional branch والتي سوف تنقل سير البرنامج إلى البايت الـ garbage وفي حالة التشغيل الطبيعي للبرنامج لا تتم هذه القفزة، لذلك لا يتم نقل سير البرنامج إلى الـ garbage ولكن في حال محاولة تشغيل البرنامج في منقح فإن هذه القفزة تتم ويتم نقل سير البرنامج إلى الـ garbage مما يؤدي إلى إنتاج كود خاطئ incorrect disassembly .

4.3.2. مثال

هذا مثال بسيط لروتين كشف المنقح عن طريق تقنية الـ PEB.BeingDebugged flag check السابق ذكرها، وتم حماية هذا الروتين بواسطة الـ anti-disassembly .

الكود الظاهر هو كود الروتين الأصلي أما باقي الكود فهو كود الـ anti-disassembly وهو يستخدم بايت الـ garbage byte 0xff ثم إضافة القفزة المشروطة fake conditional jump وذلك لإفشال عملية التحليل للكود.

```
;Anti-disassembly sequence #1
push     .jmp_real_01
stc
jnc      .jmp_fake_01
retn
.jmp_fake_01:
db       0xff
.jmp_real_01:
;--------------------------
mov      eax,dword [fs:0x18]

;Anti-disassembly sequence #2
push     .jmp_real_02
clc
jc       .jmp_fake_02
retn
.jmp_fake_02:
db       0xff
.jmp_real_02:
;--------------------------
mov      eax,dword [eax+0x30]
movzx    eax,byte [eax+0x02]
test     eax,eax
jnz      .debugger_found
```

والكود التالي يبين لنا الناتج في برنامج الـــ WinDbg :

```
0040194A    6854194000          push    0x401954
0040194F    f9                  stc
00401950    7301                jnb     image00400000+0x1953 (00401953)
00401952    c3                  ret
00401953    ff64a118            jmp     dword ptr [ecx+0x18]
00401957    0000                add     [eax],al
00401959    006864              add     [eax+0x64],ch
0040195C    194000              sbb     [eax],eax
0040195F    f8                  clc
00401960    7201                jb      image00400000+0x1963 (00401963)
00401962    c3                   ret
00401963    ff8b40300fb6        dec     dword ptr [ebx+0xb60f3040]
00401969    40                  inc     eax
0040196A    0285c0750731        add     al,[ebp+0x310775c0]
```

أما الكود التالي فهو الناتج في برنامج الـــ OllyDbg :

```
0040194A    68 54194000         PUSH 00401954
0040194F    F9                  STC
00401950    73 01               JNB SHORT 00401953
00401952    C3                  RETN
00401953    FF64A1 18           JMP DWORD PTR DS:[ECX+18]
00401957    0000                ADD BYTE PTR DS:[EAX],AL
00401959    0068 64             ADD BYTE PTR DS:[EAX+64],CH
0040195C    1940 00             SBB DWORD PTR DS:[EAX],EAX
0040195F    F8                  CLC
00401960    72 01               JB SHORT 00401963
00401962    C3                  RETN
00401963    FF8B 40300FB6       DEC DWORD PTR DS:[EBX+B60F3040]
00401969    40                  INC EAX
0040196A    0285 C0750731       ADD AL,BYTE PTR SS:[EBP+310775C0]
```

وفي النهاية هذا هو الناتج في برنامج الـــ IDAPro :

```
0040194A           push    (offset loc_401953+1)
0040194F           stc
00401950           jnb     short loc_401953
00401952           retn
00401953 ; ---------------------------------------------------------------
00401953
00401953 loc_401953:                       ; CODE XREF: sub_401946+A
00401953                                   ; DATA XREF: sub_401946+4
00401953           jmp     dword ptr [ecx+18h]
00401953 sub_401946 endp
00401953
00401953 ; ---------------------------------------------------------------
00401957           db    0
00401958           db    0
00401959           db    0
0040195A           db   68h ; h
```

```
0040195B              dd offset unk_401964
0040195F              db 0F8h ; °
00401960              db  72h ; r
00401961              db    1
00401962              db 0C3h ; +
00401963              db 0FFh
00401964 unk_401964 db  8Bh ; ï               ; DATA XREF: text:0040195B
00401965              db  40h ; @
00401966              db  30h ; 0
00401967              db  0Fh
00401968              db 0B6h ;
00401969              db  40h ; @
0040196A              db    2
0040196B              db  85h ; à
0040196C              db 0C0h ; +
0040196D              db  75h ; u
```

وستلاحظ كيف فشلت البرامج الثلاثة في تحليل الكود، وكل برنامج أظهر ناتج يختلف عن الآخر.

هذه التقنية مزعجة جدا للكراكر. أيضاً يوجد العديد من الطرق التي تستخدمها تقنية الـــ anti-disassembly وما تم ذكره هنا هو طريقة واحدة فقط. أيضاً يمكن إضافة تقنية الـــ anti-disassembly لسورس كود البرنامج كماكرو macro وبالتالي يكون السورس كود واضحاً للمبرمج.

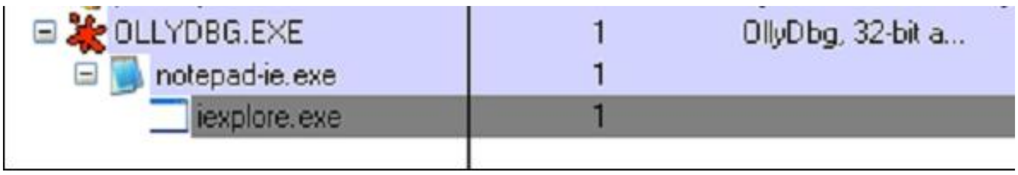
5. وسائل أخرى متقدمة ADVANCED AND OTHER TECHNIQUES

هنا سيتم ذكر بعض الطرق المتقدمة والجديدة نسبياً والتي لم يتم ذكرها في الأجزاء السابقة.

5.1. PROCESS INJECTION

5.1.1. الشرح

عملية حقن البروسيس أصبحت خياراً في بعض برامج الحماية الحديثة، ومن أشهرها Armadillo ومن المعروف أن برنامج الحماية يقوم بفك ضغط البرنامج المحمي في الذاكرة، وتعتمد هذه الطريقة على اختيار بروسيس تعمل في النظام ثم حقن البرنامج المحمي بعد فك الضغط عنه في هذا البروسيس الذي تم اختياره بغرض حمايته وإخفائه. ومن أمثلة البروسيس التي يتم الحقن فيها explorer.exe وiexplorer.exe :



أيضاً الصورة التالية تبين برنامج حماية (برنامج لتشفير السيرفرات) يدعم هذه الخاصية :



وبالتالي تستخدم السيرفرات والفيروسات هذه الطريقة للتخفي من الجدران النارية؛ لأن الجدار الناري يكون لديه قائمة بالبرامج المسموح لها بالعمل والاتصال بالانترنت، وعند حقن السيرفر داخل بروسيس خاصة ببرنامج من البرامج المسموح لها فلن يستطيع الجدار الناري كشفة وإيقاف عمله.

من الجدير بالذكر أن تعلم أن هنالك برنامج حماية عربي يستخدم هذه التقنية وهو برنامج MPack من برمجة dj-siba مدير الفريق.

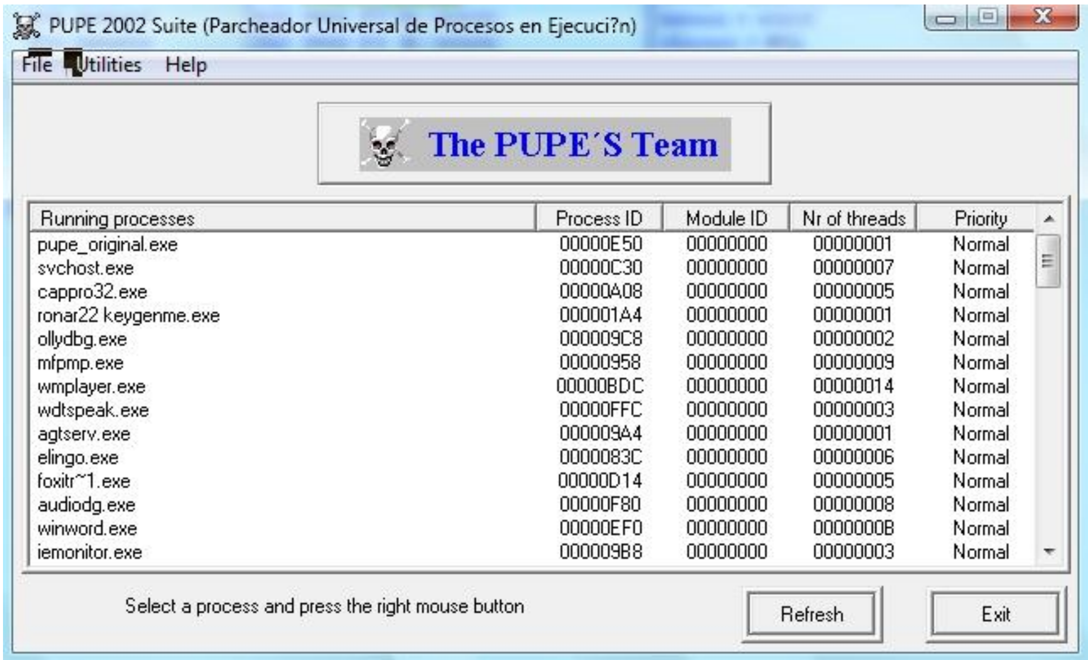
5.1.2. وسائل التغلب على هذه الطريقة

حتى تقوم بتنقيح child process ابتداءً من نقطة البداية entry point يتم ذلك عن طريق وضع نقطة توقف على الدالة WriteProcessMemory وهي المسؤولة عن كتابة الكود الخاص بالـ child process ثم نقوم بتعديل الكود ووضع حلقة غير منتهية عند نقطة البداية jump to self وفائدة هذه الحلقة الإبقاء على البروسيس الابن بعد إيقاف الــ parent process وبالتالي ستتمكن من تنقيحها وعمل dump لها من الذاكرة.

وهذا مثال لبرنامج يستخدم هذه التقنية :

```
.  68 D4344000      PUSH Ronar22_.004034D4                      ┌pProcessInfo = Ronar22_.004034D4
.  68 90344000      PUSH Ronar22_.00403490                      │pStartupInfo = Ronar22_.00403490
.  6A 00            PUSH 0                                      │CurrentDir = NULL
.  6A 00            PUSH 0                                      │pEnvironment = NULL
.  6A 04            PUSH 4                                      │CreationFlags = CREATE_SUSPENDED
.  6A 00            PUSH 0                                      │InheritHandles = FALSE
.  6A 00            PUSH 0                                      │pThreadSecurity = NULL
.  6A 00            PUSH 0                                      │pProcessSecurity = NULL
.  6A 00            PUSH 0                                      │CommandLine = NULL
.  68 00304000      PUSH Ronar22_.00403000                      │ModuleFileName = "Ronar22 Keygenme.exe"
.  E8 96000000      CALL <JMP.&kernel32.CreateProcessA>         └CreateProcessA
.  6A 00            PUSH 0                                      ┌pBytesWritten = NULL
.  68 BF010000      PUSH 1BF                                    │BytesToWrite = 1BF (447.)
.  68 B1324000      PUSH Ronar22_.004032B1                      │Buffer = Ronar22_.004032B1
.  FF35 4A304000    PUSH DWORD PTR DS:[40304A]                  │Address = 401000
.  FF35 D4344000    PUSH DWORD PTR DS:[4034D4]                  │hProcess = NULL
.  E8 97000000      CALL <JMP.&kernel32.WriteProcessMemory>     └WriteProcessMemory
.  FF35 D8344000    PUSH DWORD PTR DS:[4034D8]                  ┌hThread = NULL
.  E8 80000000      CALL <JMP.&kernel32.ResumeThread>           └ResumeThread
```

وهناك برنامج يسهل علينا عملية وضع الحلقة الغير منتهية في بداية الـــ child process يدعي PUPE 2002 :

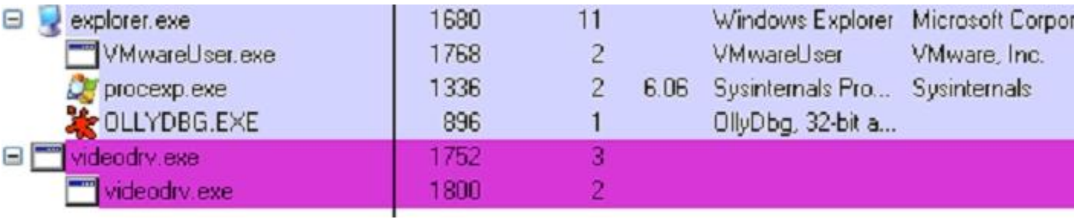


5.2. Debugger Blocker

5.2.1. الشرح

هي طريقة جديدة نسبياً تم استخدمها من قبل برنامج الحماية الشهير Armadillo وتعتمد على منع الكراكر من عمل attaching للبروسيس المحمية. وتستخدم هذه الطريقة دوال التنقيح الموجودة في الويندوز debugging functions .

تقوم فكرتها على عمل parent process هذه البروسيس تقوم بتنقيح بروسيس أخرى child process المراد حمايتها لذلك الـــ parent process يكون لها صلاحيات الـــ debugs/controls :



explorer.exe	1680	11		Windows Explorer	Microsoft Corpor
VMwareUser.exe	1768	2		VMwareUser	VMware, Inc.
procexp.exe	1336	2	6.06	Sysinternals Pro...	Sysinternals
OLLYDBG.EXE	896	1		OllyDbg, 32-bit a...	
videodrv.exe	1752	3			
videodrv.exe	1800	2			

وبما أن الــ child process المحمية يتم تنقيحها فإن أي محاولة لعمل Attach لهذه البروسيس في الديبغر سوف تفشل بسبب فشل الدالة DebugActiveProcess الموجودة في مكتبة الــ kernel32 أيضاً بسبب إرجاع الدالة الــ native المسماة NtDebugActiveProcess للقيمة STATUS_PORT_ALREADY_SET .

5.2.2. وسائل التغلب على هذه الطريقة

هنالك طريقة بسيطة للتغلب عليها، وتعتمد على محاولة استدعاء الدالة DebugActiveProcessStop في الــ context الخاص بالــ parent process وهذه الدالة سوف توقف عملية التنقيح للــ child process بعد ذلك نتمكن من عمل Attach لها في أي ديبغر.

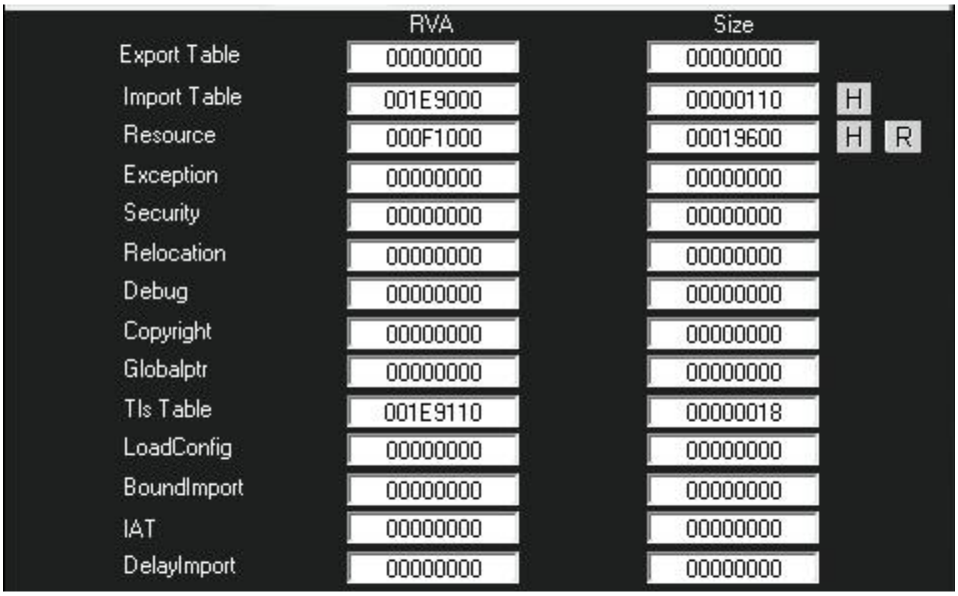
ويتم ذلك بواسطة عمل Attach للــ parent process ثم وضع نقطة توقف على الدالة WaitForDebugEvent وعند التوقف نقوم بحقن الدالة DebugActiveProcessStop(ChildProcessPID) وبعد نجاح العملية سوف نتمكن من عمل Attach للبروسيس المحمية child process .

5.3. TLS Callbacks

5.3.1. الشرح

هي وسيلة أخرى تستخدم من قبل برامج الحماية، وتعتمد على تشغيل كود قبل نقطة البداية الأصلية للبرنامج actual entry point ويتم تحقيق ذلك عن طريق الــ Thread Local Storage (TLS) . وتستغل برامج الحماية ذلك في وضع أكواد التحقق من المنقحات وأكواد فك التشفير decryption routines وبذلك لن يكون الكراكر قادراً على تنقيح هذه الأكواد.

ويتم كشف الــ TLS callbacks بواسطة برامج الــ PE Editor حيث تفحص بنية الــ Data Directory entries والتي تحتوي على العنصر TLS directory .



	RVA	Size
Export Table	00000000	00000000
Import Table	001E9000	00000110
Resource	000F1000	00019600
Exception	00000000	00000000
Security	00000000	00000000
Relocation	00000000	00000000
Debug	00000000	00000000
Copyright	00000000	00000000
Globalptr	00000000	00000000
Tls Table	001E9110	00000018
LoadConfig	00000000	00000000
BoundImport	00000000	00000000
IAT	00000000	00000000
DelayImport	00000000	00000000

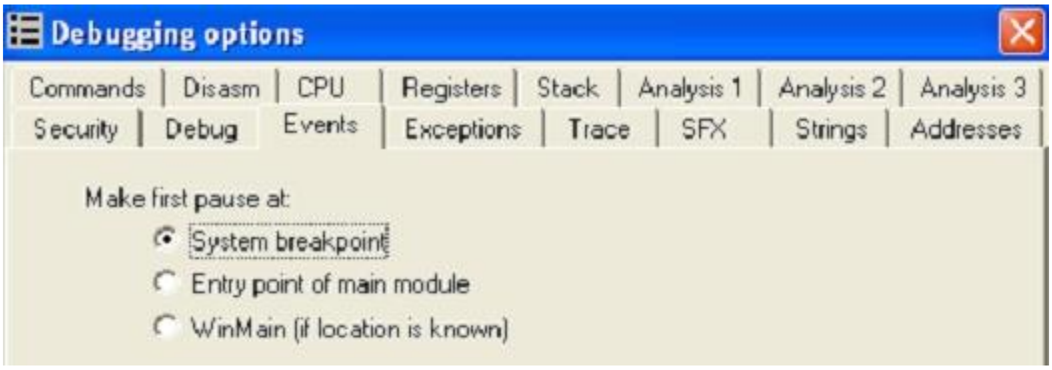
ستجد في الصورة السابقة أن الـــ RVA للـــ TLS عند 001E9110 وحجمه يساوي 18

أيضاً يمكن عرض بيانات أكثر دقة حول الـــ TLS directory :

```
TLS directory:
  StartAddressOfRawData: 00000000
  EndAddressOfRawData:   00000000
  AddressOfIndex:        004610F8
  AddressOfCallBacks:    004610FC
  SizeOfZeroFill:        00000000
  Characteristics:       00000000
```

5.3.2. وسائل التغلب على هذه الطريقة

عندما تكون الإعدادات طبيعية في برنامج OllyDbg فإنه سيقوم بالتوقف عند الـــ OEP للبرنامج المراد تنقيحه. لكن في حالة وجود الـــ TLS فإن الكود الموجود هناك سيتم تنفيذه قبل توقف OllyDbg عند الـــ OEP لذلك يتم التعديل على خيارات البرنامج ليتوقف عند الـ actual loader أي قبل تنفيذ الـــ TLS callbacks :



وبذلك سوف يتوقف OllyDbg على الـــ actual loader أي داخل مكتبة الـــ ntdll.dll وبذلك ستتمكن من وضع نقطة توقف على الـــ callback routines ومن ثم التتبع والتنقيح.



لمزيد من المعلومات حول الملفات PE ننصح بقراءة هذه المقالات :

An In-Depth Look into the Win32 Portable Executable File Format by Matt Pietrek

An In-Depth Look into the Win32 Portable Executable File Format, Part 2 by Matt Pietrek

5.4. Stolen Bytes

5.4.1. الشرح

هي عملية إزالة بعض الأكواد الهامة من البروسيس المراد حمايتها، وعادة ما تكون هذه الأكواد في الـــ EP للبروسيس والهدف هو حماية البروسيس عند عمل Dump لها من الميموري، فإن البرنامج الناتج لن يعمل. ويتم نسخ هذه الأكواد في مكان ما في الذاكرة ويتم تنفيذهم أيضاً من هذا المكان، لذلك عند عمل Dump للبروسيس فلن يتم استرجاع هذه الأكواد، لذلك تفشل عملية التشغيل.

5.4.2. المثال

المثال التالي يوضح الكود الكامل لبداية برنامج ما :

```
004011CB  MOV EAX,DWORD PTR FS:[0]
004011D1  PUSH EBP
004011D2  MOV EBP,ESP
004011D4  PUSH -1
004011D6  PUSH 0047401C
004011DB  PUSH 0040109A
004011E0  PUSH EAX
004011E1  MOV DWORD PTR FS:[0],ESP
004011E8  SUB ESP,10
004011EB  PUSH EBX
004011EC  PUSH ESI
004011ED  PUSH EDI
```

المثال التالي هو نفس الكود لكن بعد حمايته بـــ Enigma Protector وسرقة أول تعليمتين في البرنامج :

```
004011CB   POP EBX
004011CC   CMP EBX,EBX
004011CE   DEC ESP
004011CF   POP ES
004011D0   JECXZ SHORT 00401169
004011D2   MOV EBP,ESP
004011D4   PUSH -1
004011D6   PUSH 0047401C
004011DB   PUSH 0040109A
004011E0   PUSH EAX
004011E1   MOV DWORD PTR FS:[0],ESP
004011E8   SUB ESP,10
004011EB   PUSH EBX
004011EC   PUSH ESI
004011ED   PUSH EDI
```

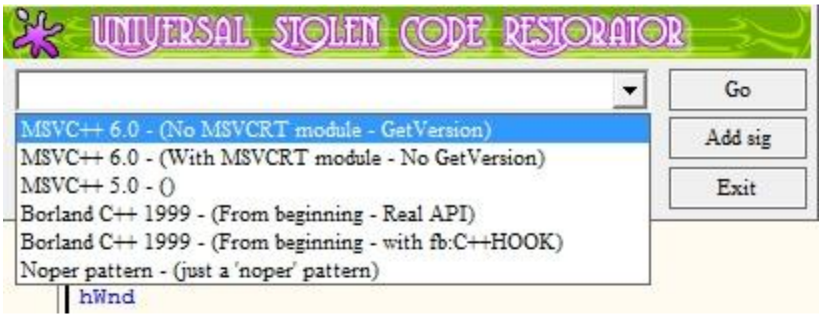
وهذا أيضاً مثال لبرنامج محمي بواسطة الـــ ASProtect حيث يقوم بإضافة JUMP إلى مكان ما لتنفيذ الـــ stolen instructions وطبعاً يتم حماية الـــ stolen instructions بوضع garbage code خلالها.

```
004011CB   JMP 00B70361
004011D0   JNO SHORT 00401198
004011D3   INC EBX
004011D4   ADC AL,0B3
004011D6   JL SHORT 00401196
004011D8   INT1
004011D9   LAHF
004011DA   PUSHFD
004011DB   MOV EBX,1D0F0294
004011E0   PUSH ES
004011E1   MOV EBX,A732F973
004011E6   ADC BYTE PTR DS:[EDX-E],CH
004011E9   MOV ECX,EBP
004011EB   DAS
004011EC   DAA
004011ED   AND DWORD PTR DS:[EBX+58BA76D7],ECX
```

5.4.3. وسائل التغلب على هذه الطريقة

دائماً تكون الأكواد المسروقة من البرنامج هي أكود في الـــ OEP للبرنامج، وهذا يسهل عملية الاسترجاع، وذلك بمقارنة البرنامج مع برنامج آخر من نفس الكومبيلر، وبذلك نستطيع معرفة الأكواد المسروقة من البرنامج وإعادة كتابتها.

وهناك أداة عبارة عن Plugin لبرنامج OllyDbg من برمجة Zool@nder سهلت عملية استرجاع الكود كثيراً وهي : IDAFicator



بعد اختيار الكومبيلر سيتم كتابة الكود الخاص به، بعد ذلك تستطيع إدخاله إلى البرنامج بضغطة زر واحدة :

```
$  55                  PUSH EBP
.  8BEC                MOV EBP,ESP
.  6A FF               PUSH -1
.  68 DEADC0DE         PUSH DEC0ADDE
.  68 DEADC0DE         PUSH DEC0ADDE                      SE handler installation
.  64:A1 00000000      MOV EAX,DWORD PTR FS:[0]
.  50                  PUSH EAX
.  64:8925 00000000    MOV DWORD PTR FS:[0],ESP
.  83EC 58             SUB ESP,58
.  53                  PUSH EBX
.  56                  PUSH ESI
.  57                  PUSH EDI
.  8965 E8             MOV DWORD PTR SS:[EBP-18],ESP
```

5.5. API Redirection

5.5.1. الشرح

من أشهر طرق حماية البرامج وهي تمنع الكراكر من استعادة جدول الدوال المستوردة الخاصة بالبرنامج بسهولة rebuilding the import table .

يتم في هذه الطريقة تدمير جدول الدوال والاستدعاءات للدوال الأصلية، حيث يتم عمل إعادة توجيه إلى روتين في الذاكرة، وهذا الروتين يكون مسؤولاً فيما بعد عن استدعاء الدالة الأصلية.

هذا المثال يوضح كود لاستدعاء الدالة CopyFileA :

```
00404F05   LEA EDI,DWORD PTR SS:[EBP-20C]
00404F0B   PUSH EDI
00404F0C   PUSH DWORD PTR SS:[EBP-210]
00404F12   CALL <JMP.&KERNEL32.CopyFileA>
```

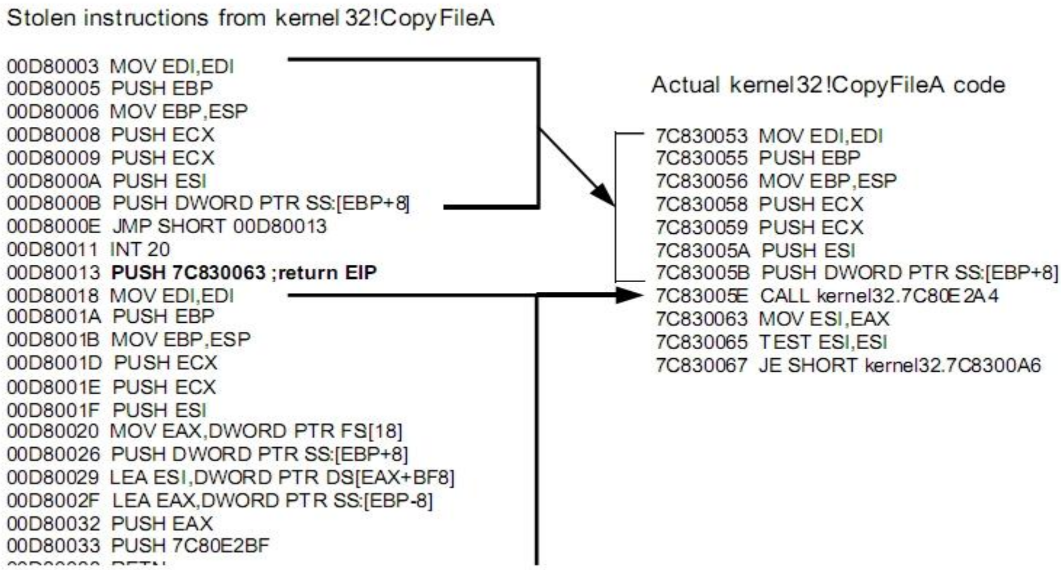
أمر النداء هذا ما هو إلا أمر قفز إلى عنوان الدالة في الـــ Import Table :

```
004056B8   JMP DWORD PTR DS:[<&KERNEL32.CopyFileA>]
```

وعند أخذ مثال لحماية Asprotect ستجد أن أمر النداء في المثال السابق تحول إلى آمر نداء إلى روتين في الذاكرة allocated memory وهذا الروتين يحتوي على تعليمات مسروقة من الـــ kernel32!CopyFileA :

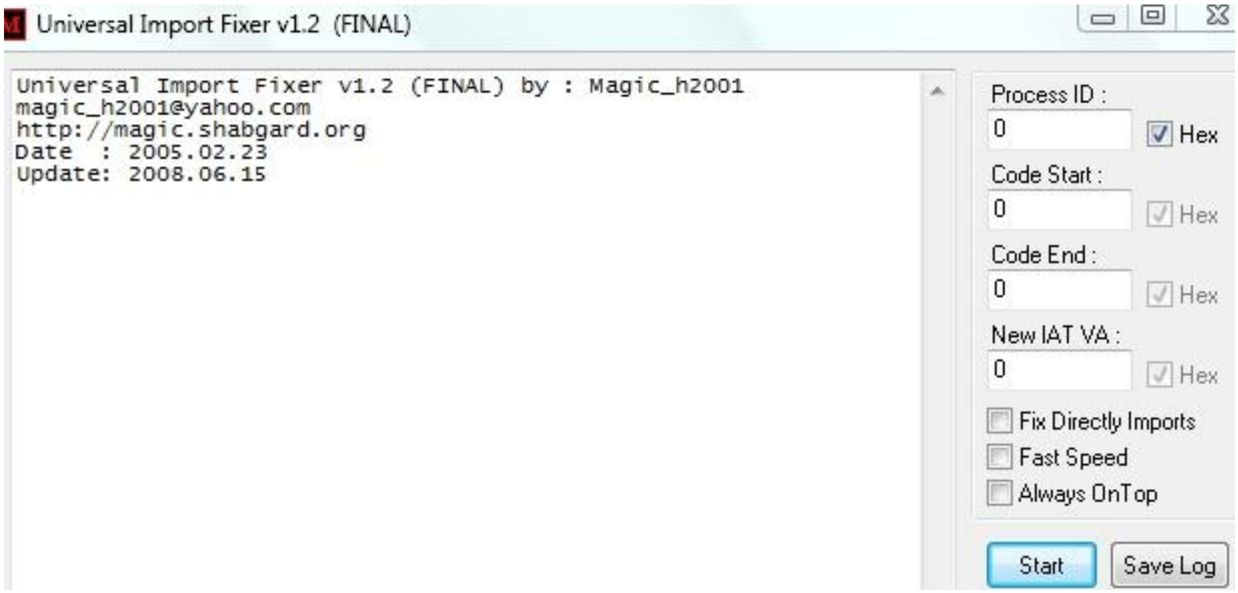
```
004056B8   CALL 00D90000
```

والصورة التالية تبين خطوات كتابة التعليمات المسروقة، فإذا نظرنا للمثال السابق سنجد أن أول 7 تعليمات من الدالة kernel!CopyFileA قد تم أخذها بالإضافة إلى أمر النداء عند العنوان 7C83005E بعد ذلك يتم نقل سير البرنامج إلى داخل مكتبة الـــ kernel32.dll بالتحديد في منتصف الدالة CopyFileA عن طريق RETN to 0x7C830063 :



5.5.2. وسائل التغلب على هذه الطريقة

هذه الطريقة بالتحديد تحتاج إلى دراسة مسفيضة لبرنامج الحماية لمعرفة متى وأين يقوم بعمل Redirecting للدوال؟ أيضاً هناك بعض البرامج التي تستخدم لإصلاح الدوال وتصليح جدول الـــ Import Table مثل برنامج الـــ UIF :



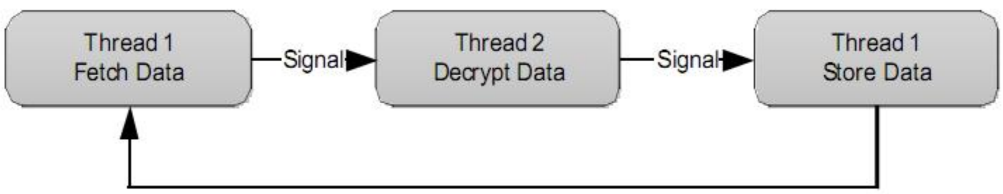
5.6 Multi-Threaded Packers

5.6.1 الشرح

هذه الطريقة تعتمد على إنتاج ثريد آخر من بروسيس الخاصة ببرنامج الحماية، هذا الثريد يتم فيه عمل العمليات الهامة؛ مثل عمليات فك التشفير.

مثال على ذلك برنامج PECrypt حيث يقوم باستخدام ثريد ثانية للقيام بعملية فك التشفير عن الكود الذي تم استخراجه من الثريد الرئيسية fetched by the main thread :

PECrypt operates and synchronizes its threads as follows:

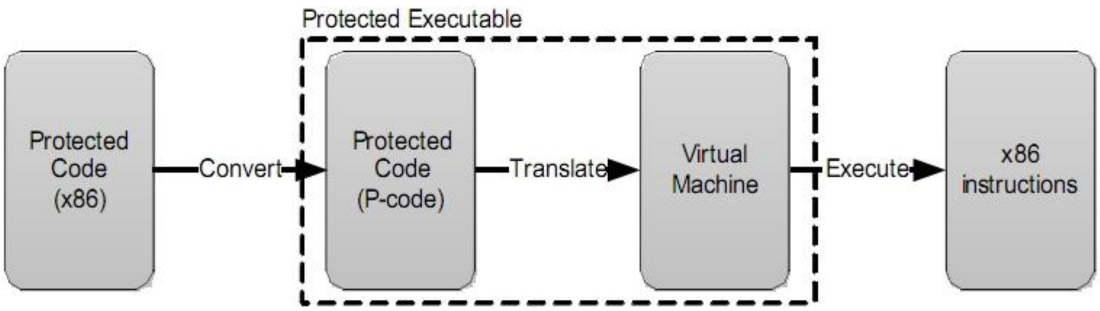


5.7 Virtual Machines

5.7.1 الشرح

سبب استخدام هذه التقنية بسيط وهو أن الكراكر دائماً ما يتغلبون على طرق الـ anti-debugging وطرق الـ Anti reversing وهي الطرق التي ذكرناها سابقاً، فكما لاحظتم أنه بعد كل طريقة يتم وضع وسيلة للتغلب عليها.

لذلك قامت شركات حماية البرامج الكبرى بابتكار هذه الطريقة، والتي تعتمد على ترجمة الأجزاء المحمية من الكود إلى p-codes والذي يتم ترجمته بعد ذلك إلى لغة الآلة machine code ليتم تنفيذه. وبذلك تم مضاعفة حماية الكود وجعل عملية فهم ما يقوم به كود البرنامج أمراً صعباً جداً.



وتستخدم برامج الحماية الكبرى أمثال Oreans technologies' CodeVirtualizer و StarForce و Execrypt تقنية الـ Virtual Machines .

5.7.2. وسائل التغلب على هذه الطريقة

التغلب على هذه التقنية ليس بالأمر السهل، بل يتوجب على الكراكر دراسة كيفية توليد الـ p-code وكيفية ترجمته بواسطة الـ virtual machine وبواسطة المعلومات التي يحصل عليها الكراكر يمكن تطوير الـ disassemble الذي يقوم بعملية التحويل والترجمة.

ولمزيد من المعلومات حول تطوير الـ p-code disassemble وكيفية عمل الـ virtual machines يرجى قراءة هذا المقال :

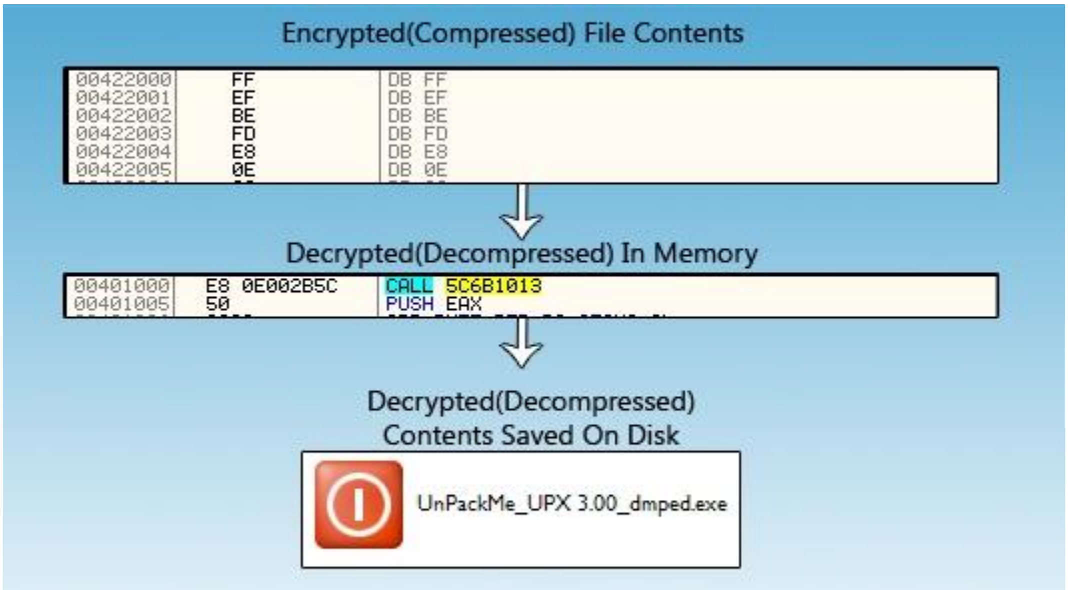
Defeating HyperUnpackMe2 With an IDA

من الجدير بالذكر أن كل البرامج التي استخدمت تقنية الـ virtual machine قد تم كسرها ويوجد في الانترنت مئات المقالات وبرامج فك الضغط والاسكريبت التي تمكننا من كسر أي برنامج حماية على وجه الأرض.

5.8. Anti-Dumping

5.8.1. الشرح

كما تحدثنا من قبل فإن برامج الحماية تقوم بفك تشفير كود البرنامج المراد حمايته في الذاكرة، وذلك لكي يعمل البرنامج المحمي. لذلك كان يجب ابتكار وسائل تمنع الكراكر من تنفيذ هذه العملية، وهي عملية الـــ Dump أي أخذ نسخة من البرنامج بعد فك تشفيره في الذاكرة وحفظ هذه النسخة على الهارد ديسك.



هناك طرق سبق ذكرها ممكن أن تدخل تحت نظام الـــ Anti-Dumping مثل الـــ Virtual Machines والـــ Stolen Bytes وسوف نعرض سريعاُ بعض الطرق الأخرى :

طريقة تسمى Nanomites وهي عبارة عن استبدال العديد من أوامر القفز في البرنامج branch instructions بأوامر أخرى خاصة بنقاط التوقف INT 3 ثم حفظ اماكن هذه القفزات في جدول يتم تشفيره جيداً. ونقصد بحفظ أماكن القفزات أي حفظ المكان الذي كانت ستقفز إليه destination of the jump وكذلك رايات المعالج وطول التعليمة size of the jump بعد ذلك يقوم برنامج الحماية بعمل عملية تنقيح ذاتي، وعند حدوث أي نقطة توقف من النقاط التي تكونت نتيجة استبدال التعليمات بـــ INT 3 يقوم البرنامج بالرجوع إلى الجدول وإرجاع تعليمة القفز الأصلية. ولزيادة التعقيد يتم وضع نقاط توقف بشكل عشوائي، وهذه النقاط لا علاقة لها بالقفزات، لكنها لتضليل الكراكر. وأشهر برامج الحماية التي تستخدم هذه الطريقة برنامج الـــ Armadillo .

زيادة حجم الـ SizeOfImage وهو عنصر داخل بنية IMAGE_OPTION_HEADER الخاصة ببنية الملفات PE التي تعمل على الويندوز وبزيادة هذه القيمة في الـ PEB في الـ runtime يؤدي إلى حدوث مشاكل مع البرامج التي لم تضع حلاً لتفادي ذلك :

```
mov eax, fs:[0x30] ;PEB
mov eax, [eax + 0x0c] ;PEB_LDR_DATA
mov eax, [eax + 0x0c] ;InOrderModuleList
mov dword ptr [eax + 0x20], NewSize ;SizeOfImage
```

بخلاف الطريقة السابقة فإنه يمكن حذف كل بيانات الـ PE أي حذف الـ Portable Executable Header وبذلك سيفقد البرنامج الكثير من المعلومات الهامة؛ مثل الـ RVA للكثير من الجداول الهامة؛ مثل Reloc, Import, Export أيضاً حذف عنوان الـ entry point وأيضاً الكثير من البيانات التي يحتاجها لودر الويندوز لكي يقوم بتشغيل الملف. وبالتالي فعند عمل Dump للملف المحمي من الذاكرة سيستحيل أن يعمل. وهذا كود يبين طريقة حذف الـ PE Header :

```
// This function will erase the current images
// PE header from memory preventing a successful image
// if dumped
inline void ErasePEHeaderFromMemory()
{
    DWORD OldProtect = 0;
    // Get base address of module
    char *pBaseAddr = (char*)GetModuleHandle(NULL);
    // Change memory protection
    VirtualProtect(pBaseAddr, 4096, // Assume x86 page size
            PAGE_READWRITE, &OldProtect);
    // Erase the header
    ZeroMemory(pBaseAddr, 4096);
}
```

6. مراجع هامة

1) An Anti-Reverse Engineering Guide (By Josh_Jackson)

2) The Art of Unpacking by Mark Vincent Yason

3) Peter Ferrie collection:
   - Anti-Unpacker Tricks 2 Part Six
   - Anti-Unpacker Tricks 2 Part Five
   - Anti-Unpacker Tricks 2 Part Four
   - Anti-Unpacker Tricks 2 Part Three
   - Anti-Unpacker Tricks 2 Part Two
   - Anti-Unpacker Tricks 2 Part One

4) Reversing: Secrets of Reverse Engineering. E.Eilam. Wiley, 2005

5) OpenRCE Anti Reverse Engineering Techniques Database

6) http://www.datasecurity-event.com/uploads/unpackers.pdf

الباب الخامس: تحليل برمجيات دوت نت

تحليل تطبيقات الدوت نت [1]

.Net Applications Analysis (1)

تجدر الإشارة إلى أن هناك تجميعة لجميع ما يحتاجه المحلل العكسي من أدوات للتعامل مع الدووتنيت جعلها أخونا RobenhoodArab في pack تجده في هذه المشاركة و إلا فيمكنك الإكتفاء بمرفقات الدروس لأنها كافية لتطبيق دروس هذا الكتاب.

الدرس سيكون على الشكل التالي :

1) مقدمة عن تطبيقات الدوت نت
2) تلميحات عن برنامج Reflector
3) مقدمة عن الإضافة Deblector
4) تحليل البرنامج الهدف وإيجاد السريال.
5) احتمالات السريالات ضمن الكيجن.

- Reflector
- Deblector

DotNet EXPERT 2.exe

I. أولاً: مقدمة عن تطبيقات الدوت نت Introduction to .NET Applications

تعتمد لغات البرمجة مثل Visual Basic و C++ على مترجم خاص Compiler وظيفته تحويل الشفرة المصدرية إلى ملفات تنفيذية. ولتشغيل هذه الملفات تحتاج إلى بيئة تشغيل Runtime خاص باللغة البرمجية التي أنتجتها. وهذه البيئة تختلف باختلاف نوع اللغة البرمجية.

أما مع الحزمة الجديدة من لغات البرمجة التي تعمل ضمن بيئة الدوت نت, فقد أصبح الأمر مختلفاً, فهي تتبع المنهج التالي في إنتاج وتشغيل تطبيقات الدوت نت :

1) يحول المترجم الخاص بدوت نت Compiler الشيفرة المصدرية إلى لغة وسيطة Intermediate Language ويرمز لها اختصاراً بـــ (IL) وينتج عنه ملف تنفيذي يحتوي على تعليمات اللغة الوسيطة.

2) عند تشغيل الملف التنفيذي السابق, تتولى منصة التشغيل العامة Common Language Runtime والتي يرمز لها اختصاراً (CLR) والمضمنة ضمن إطار عمل دوت نت net framework تحويل تعليمات اللغة الوسيطة إلى لغة الآلة Machine Code .

- لاحظ هنا أن البرنامج إلى الآن لم ينتقل إلى مرحلة التنفيذ، وتعرف المرحلة السابقة بـــ Just In Time Compile واختصاراً (JIT Compiler) .

3) بعد ذلك يتسلم النظام تعليمات لغة الآلة وينفذها، وبالتالي يتم تشغيل البرنامج على مرحلتين:

- المرحلة الأولى : تحويل تعليمات اللغة الوسيطة إلى لغة الآلة.
- المرحلة الثانية : تنفيذ تعليمات لغة الآلة.

و هذا ما يفسر تأخر تنفيذ التطبيقات المبرمجة بدوت نت.

إذنً ما هي تعليمات اللغة الوسيطة IL ؟

أرفقنا مع هذا الدرس مقالة للأخ kurapica يسرد فيها تعليمات هذه اللغة .

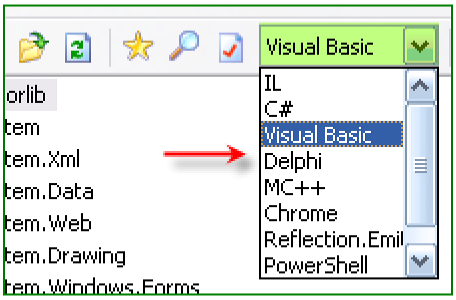
2. ثانياً: تلميحات عن برنامج Reflector

يعتبر برنامج Reflector من أفضل البرامج التي تفكك وتعرض لك محتويات تطبيقات الدوت نت, وحقيقة يعتبر عرض الكود بعدة لغات برمجية ميزة رائعة يتمتع بها هذا المفكك, وتزيد من قوته عدد من الإضافات البرمجية أبرزها : Deblector و Reflexil التي ينتظرها مستقبلاً مشرقاً في مجالها.

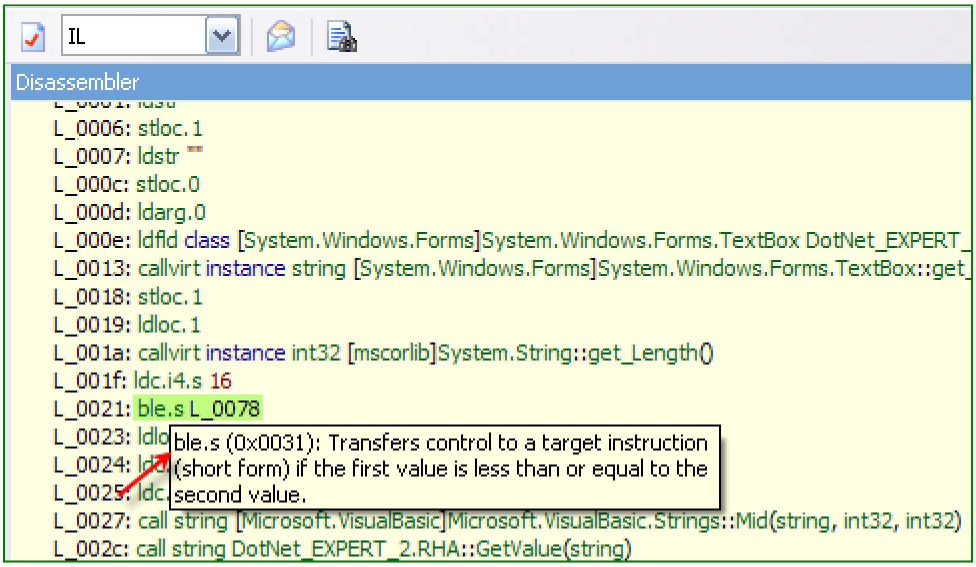
وسنلقي نظرة على أهم النقاط والميزات التي يتمتع بها البرنامج.

2.1 اختيار لغة البرمجة

تسمح لك القائمة المنسدلة في الأعلى باختيار اللغة البرمجية التي تتقنها ليسهل عليك فهم الكود.



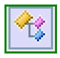
ولكن عند اختيارك للخيار (IL) ستشاهد تلميحات تشرح عمل كل تعليمة في اللغة الوسيطة وهذا الأمر يعتبر مساعدة جيدة للمبتدئين في تعلم تعليمات اللغة الوسيطة (IL) .



2.2 خيارات البحث

عند الضغط على زر البحث  سيظهر لك مربع البحث ويحتوي بجانبه على أربع أزرار تقوم بعملية توجيه وفلترة البحث.



 : يفيد هذا الخيار في البحث عن نوع كائن Object معين وفضاءات الأسماء التي تحتويه, مثلاً : ابحث عن (progressbar) .

 : يفيد هذا الخيار بالبحث عن اسم عنصر معين؛ مثلاً: في برنامجنا الهدف : ابحث عن (BTR) .

 : يفيد هذا الخيار في البحث عن سلسلة نصية ضمن الكود , مثلاً : ابحث عن (NameSpace) .

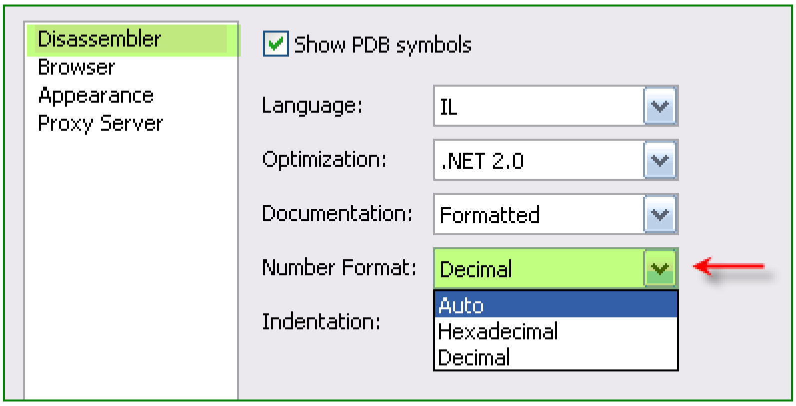
 : يستخدم هذا الخيار مترافقاً مع الخيارات الثلاث السابقة، ويستخدم للبحث عن النص المطابق تماماً، ويستبعد النصوص التي تحوي هذه الكلمة جزئياً؛ مثلاً : عند البحث عن BTR سيخرج لك عدداً كبيراً من النتائج، ولكن عند استخدام هذا الخيار ستتقلص النتائج إلى نتيجة واحدة فقط.

2.3 خيارات البرنامج

عند الضغط على زر الخيارات Options  تخرج لنا نافذة جديدة تحتوي على مجموعة من الصفحات :

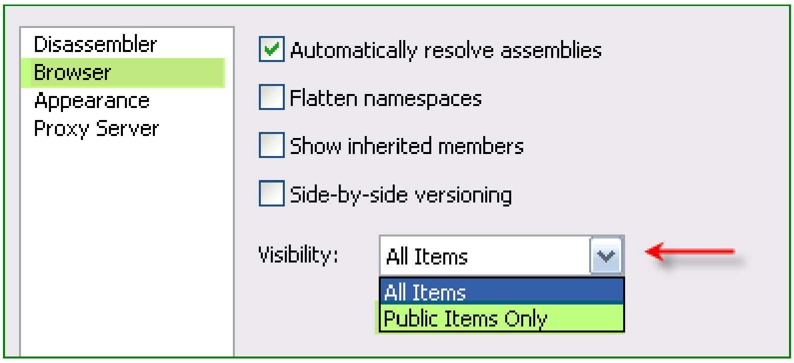
2.3.1 الصفحة Disassembler

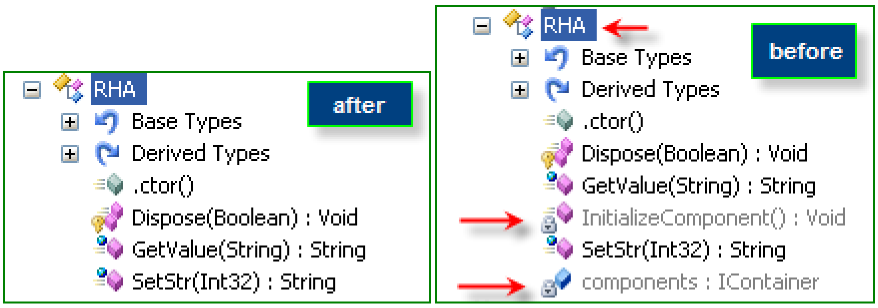
إعدادات عامة وخيار Number Format الذي يتيح عرض الأرقام بالنظام العشري أو الستة عشري.



2.3.2 الصفحة Browser

يمكننا تقليل عدد العناصر الظاهرة بتغيير الخيار Visibility إلى Public Items Only :

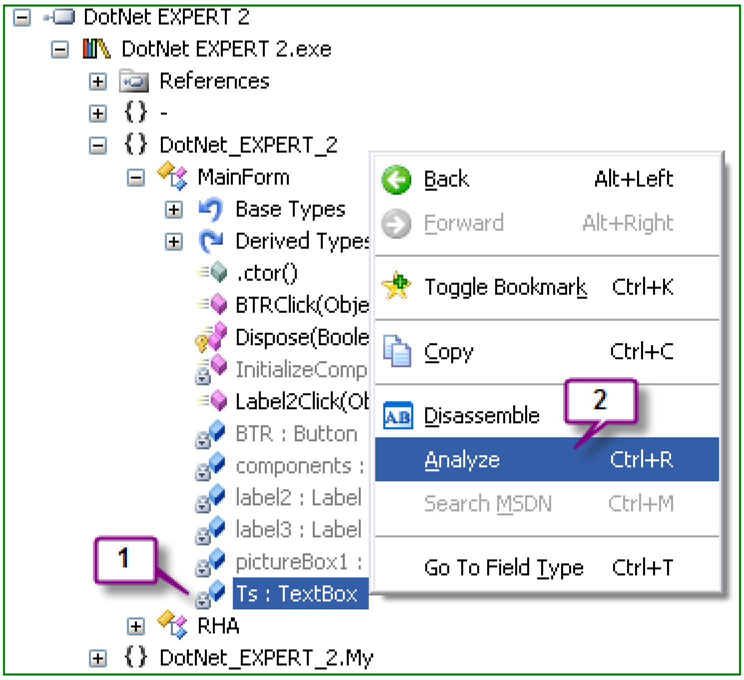




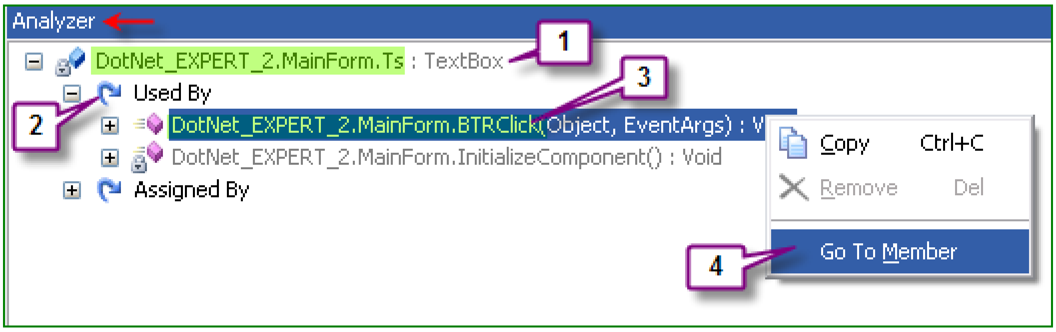
2.4 الميزة Go To Member .. Used By .. Analyze

تعتبر هذه التسلسلية من التعليمات المفيدة جداً في معرفة مكان استخدام العنصر ومكان إسناد القيم له. لنرى مثالاً على ذلك أسقط البرنامج الهدف ضمن برنامج Reflector :

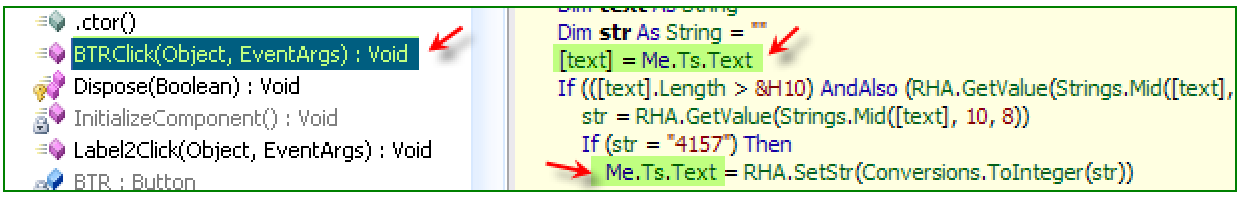
– سنحاول هنا معرفة أين يتم استخدام المربع النصي TS وأين يتم إسناد قيمة أو خاصية له؟ عند الضغط بالزر الأيمن للفأرة عليه نختار من القائمة Analyze :



ستظهر على الفور نافذة جديدة بعنوان: Analyzer تحوي ضمنها مكان استخدام العنصر وعند توسيع القائمة (+) سنجد القائمة الفرعية (Used By) ونلاحظ أن العنصر يستخدم في الحدث BTRCLICK وللذهاب إلى هذا الحدث وعرض الكود نختار الأمر Go To Member من القائمة المنبثقة (Popup Menu) وسنذهب إلى العنصر ضمن اللوحة Browse :



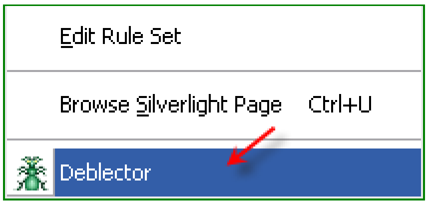
عند الضغط المزدوج عليه سيخرج لنا الكود الموافق :



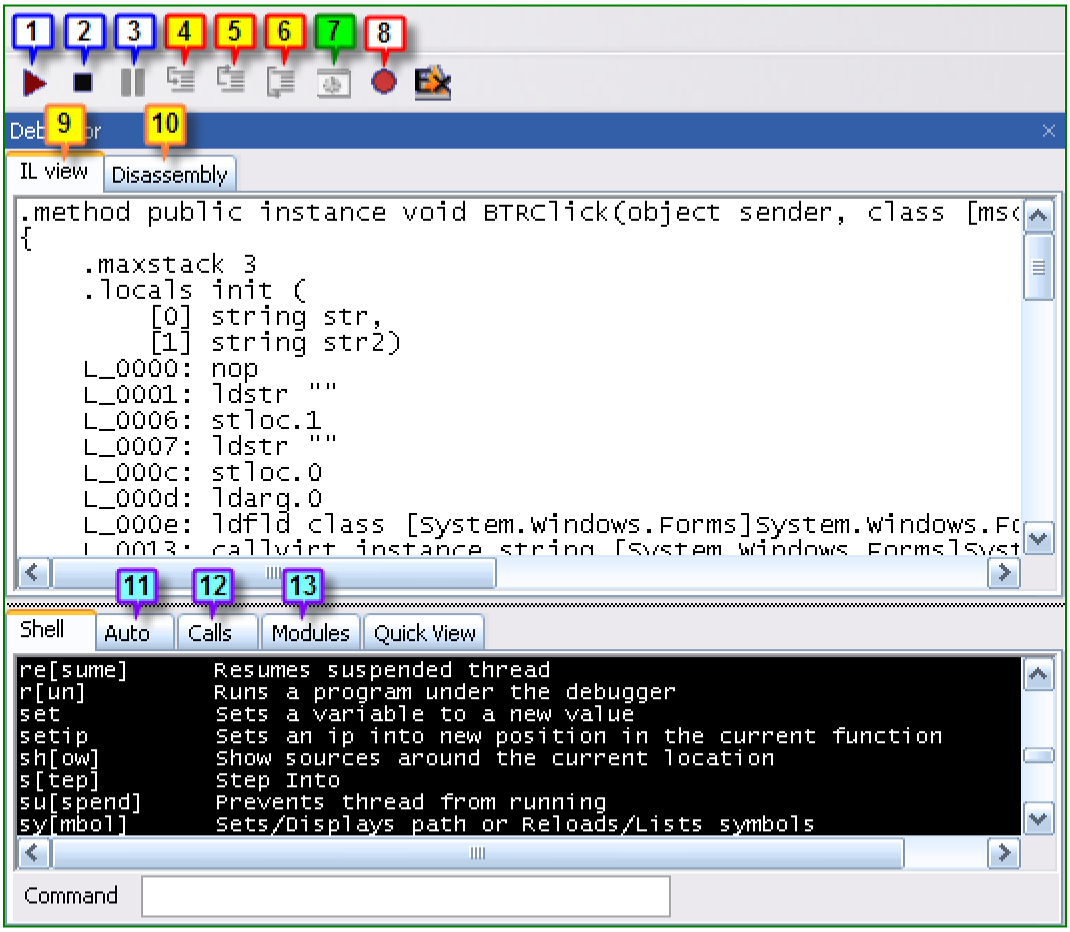
3. ثالثاً: مقدمة عن الإضافة Deblector

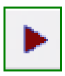
تعتبر هذه الإضافة بمثابة منقح لتطبيقات الدوت نت، ولكنها لم تصل إلى حد من التطور لتصل لمستوى OllyDbg ولكنها خيار رائع لتنقيح برمجيات الدوت نت. والشيء المهم لإتقان التعامل معها هي المعرفة الجيدة بتعليمات اللغة الوسيطة IL .

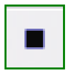
– يمكننا الوصول لهذه الإضافة من خلال القائمة Tools :



– بعد اختيارها ستفتح لك نوافذ المنقح ويضاف إلى قوائم البرنامج Reflector القائمة Debug كما ستضاف مجموعة من الأزرار :



 : تشغيل البرنامج بعد تحميله ضمن المنقح واختصاره [F7].

 : إلغاء تحميل البرنامج ضمن المنقح وإغلاق الملف التنفيذي.

 : إيقاف مؤقت لتنفيذ البرنامج، ويفيد في وضع نقاط التوقف.

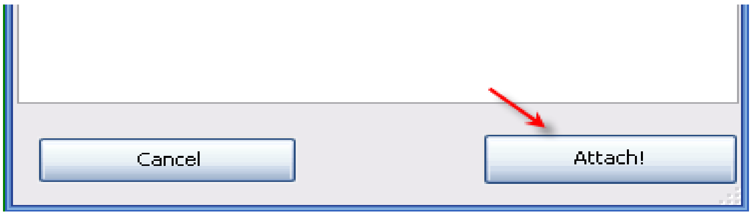
 : الدخول ضمن الاستدعاءات (Step Into) خلال التتبع واختصاره [F11].

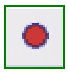
 : الخروج من الاستدعاء الحالي (Step Out) خلال التتبع.

 : تنفيذ الاستدعاءات دون الدخول إليها (Step Over) خلال التتبع واختصاره [F10].

 : تحميل الملفات التنفيذية المشغلة (المحملة في الذاكرة) ضمن المنقح (Attach) وهو المرحلة الأولى للبدء بالتنقيح .

عند اختياره تفتح لك نافذة تسمح لك باختيار الملف الذي تريد تنقيحه :



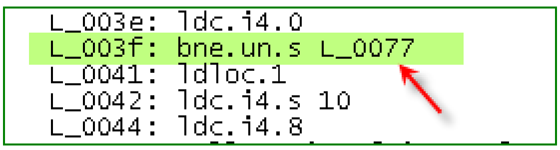
 : تساعد في وضع نقاط التوقف في مكان المؤشر ضمن النافذة IL View واختصاره [F9].

و يجب الانتباه إلى أنه لا يمكن وضع نقطة توقف إلا إذا كان المنقح في حال توقف مؤقت  .

3.1 النافذة IL View

تعرض تعليمات البرنامج باللغة الوسيطة IL ويمثل الترتيب L_0001 ... L_0000 : رقم السطر، كما يلاحظ وجود هذه الترتيب خلف تعليمات القفز ويشير إلى الموقع الذي سيتم نقل التنفيذ إليه.

فمثلاً هنا : إذا كانت القيمتين غير متساويتين (bne) سيتم القفز إلى السطر L_0077 :



```
L_003e: ldc.i4.0
L_003f: bne.un.s L_0077
L_0041: ldloc.1
L_0042: ldc.i4.s 10
L_0044: ldc.i4.8
```

3.2 النافذة DISASSEMBLY

تعرض الكود المصدري الموافق لما يوجد من تعليمات اللغة الوسيطة الموجودة في النافذة IL View .

3.3 النافذة AUTO

تعتبر نافذة مهمة جداً ويتم فيها عرض قيم المتحولات والقيم المسندة للعناصر أثناء التنقيح.

3.4 النافذة MODUELS

تحتوي على الملفات المحملة اللازمة لتشغيل البرنامج. كما تظهر الملفات التي يقوم بتحميلها البرنامج كملفات DLL وهي مفيدة في حال أردنا معرفة موارد البرنامج.

4. رابعاً: تحليل البرنامج الهدف وإيجاد السريال

إذا تصفحنا كود الإجراء BTRCLICK الذي يمثل الضغط على الزر Register :

Disassembler

```
Public Sub BTRClick(ByVal sender As Object, ByVal e As EventArgs)
    Dim text As String = ""
    Dim str As String = ""
    [text] = Me.Ts.Text
    If (([text].Length > 16) AndAlso (RHA.GetValue(Strings.Mid([text], 1, 9)) = "112244617383124392")) Then
        str = RHA.GetValue(Strings.Mid([text], 10, 8))
        If (str = "4157") Then
            Me.Ts.Text = RHA.SetStr(Conversions.ToInteger(str))
        End If
    End If
End Sub
```

المتغير Text : متغير نصي String, يسند له النص المدخل الموجود في المربع النصي TS .

Text.Length > 16 : يتم التأكد من أن النص المدخل أكبر من 16 حرفاً، أي 17 أو أكثر.

Mid(text, 1, 9) : أخذ أول 9 محارف من السريال المدخل.

Mid(text, 10, 8) : أخذ 8 محارف من السريال المدخل اعتبارا من المحرف العاشر.

RHA.GetValue : GetValue وظيفة Function موجودة في الوحدة البرمجية RHA.

Ts.text : إسناد قيمة نصية للمربع النصي Ts.

9.1. التحليل الأولي

يتم أخذ السريال المدخل.

يتم التأكد من أن السريال المدخل مؤلف من أكثر من 16 حرف.

يتم تمرير أول 9 حروف من السريال إلى الدالة GetValue ومقارنة ما تعيده الدالة مع الرقم (112244617383124392).

إذا تحققت المساواة ننتقل إلى المساواة التالية.

يتم تمرير الثمانية أرقام التالية من السريال إلى الدالة GetValue ومقارنة ما تعيده الدالة مع الرقم (4157).

إذا كانت المساواة صحيحة يتم إسناد قيمة نصية للمربع TS بمساعدة الدالة SetStr .

9.2. تحليل الدالة GetValue

```
Public Shared Function GetValue(ByVal StrIn As String) As String
    Dim str9 As String = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-1234567890123456789"
    Dim str4 As String = ""
    Dim str5 As String = ""
```

نلاحظ في البداية تعريف الدالة ونلاحظ أن ما يدخل إلى الدالة (Parameter) يجب أن يكون من النوع النصي String ويرمز له بـــ StrIn والقيمة التي تعيدها الدالة هي من النوع النصي، أيضاً ويتم التصريح عنها بـــ GetValue = ...

```
Public Shared Function GetValue(ByVal StrIn As String) As String
    Dim str9 As String = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-1234567890123456789"
    Dim str4 As String = ""
    Dim str5 As String = ""
    Dim str6 As String = ""
    Dim str7 As String = ""
    Dim str8 As String = ""
    Dim str2 As String = ""
    Dim str3 As String = ""
    Dim start As Integer = 1
    Dim num As Integer = 0
    Do
      If ((num Mod 4) = 0) Then
        str4 = (str4 & Conversions.ToString(str9.Chars(num)))
      ElseIf ((num Mod 4) = 1) Then
        str5 = (str5 & Conversions.ToString(str9.Chars(num)))
      ElseIf ((num Mod 4) = 2) Then
        str6 = (str6 & Conversions.ToString(str9.Chars(num)))
      ElseIf ((num Mod 4) = 3) Then
        str7 = (str7 & Conversions.ToString(str9.Chars(num)))
      End If
      num += 1
    Loop While (num <= 71)
```

– كما تلاحظ فهنالك جملة من المتغيرات النصية :

Str9 : يمثل متغير نصي ثابت القيمة (مؤلف من 72 محرفاً).

Str4, Str5, Str6, Str7 : متغيرات نصية.

– لاحظ أن هذه المتحولات النصية يتم إسناد القيم لها من خلال الحلقة الظاهرة في الأعلى Loop ... Do ويلعب المتحول الرقمي num دور العداد لهذه الحلقة فهو يبدأ بالقيمة (0) وتنتهي الحلقة عندما تصبح قيمته (71) أي سيتم تكرار هذه الحلقة (72) مرة.

– كما نلاحظ أن هناك أربع جمل شرطية :

If (num Mod 4) = 0 Then .... وهذه تعني أنه إذا كان باقي قسمة العدد num على 4 يساوي (0) عندئذٍ = Str4 Str4 + Str9 [num] وهذا يعني أن قيمة المتغير Str4 تساوي قيمته الحالية مضاف إليها الحرف الذي ترتيبه قيمة num ضمن السلسلة النصية Str9 :

Num = 0 ➔ 0 Mod 4 = 0 ➔ Str4 = Str4 + Str9(0) ➔ Str4 = a

Num = 1 ➔ 1 Mod 4 = 1 ➔ St5 = Str5 + Str9(1) ➔ Str5 = b

– ونتيجة تكرار هذه الحلقة 72 مرة سينتج لدينا أربع سلاسل نصية (**Str4, Str5, Str6, Str7**) ولكن كيف سنعرف قيمهم ؟ سنستخدم المنقح لتتبع قيم هذه السلاسل، ولنتبع الخطوات التالية :

1) افتح الإضافة Deblector.

2) شغل البرنامج الهدف (DotNet Expert 2.exe).

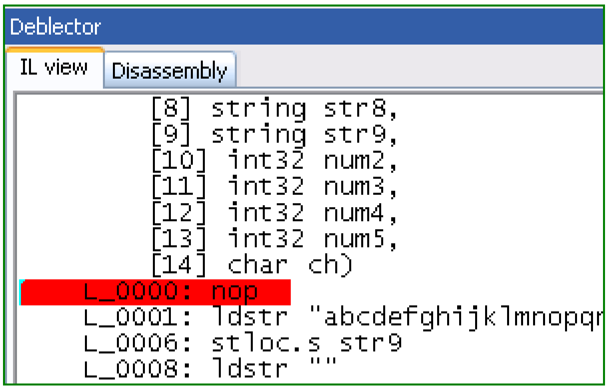
3) اضغط على زر Attach  .

4) اختر البرنامج الهدف من النافذة ثم اضغط Attach.



5) توجه إلى كود الدالة GetValue الموجودة في الوحدة RHA .

6) ضع نقطة توقف عند السطر L_0000 : nop (بوضع المؤشر عندها والضغط على F9 أو الزر) سيتحول لون السطر إلى اللون الأحمر.

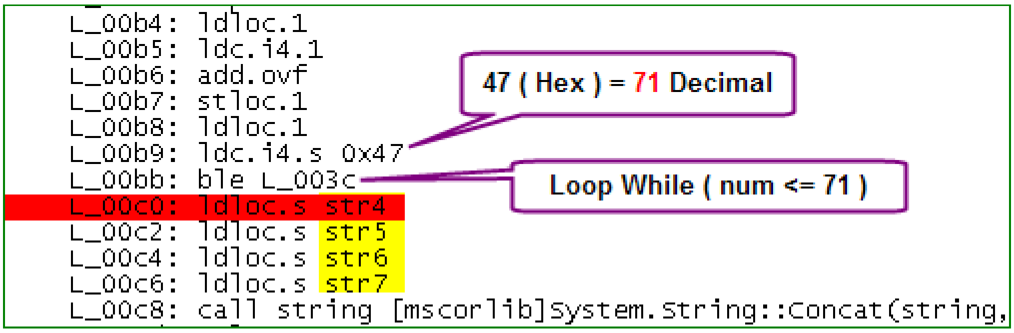


7) شغل البرنامج بالضغط على F7 أو أضغط على الزر .

8) أدخل رقم التسجيل التالي (طوله أكثر من 16) : 112233445566778899

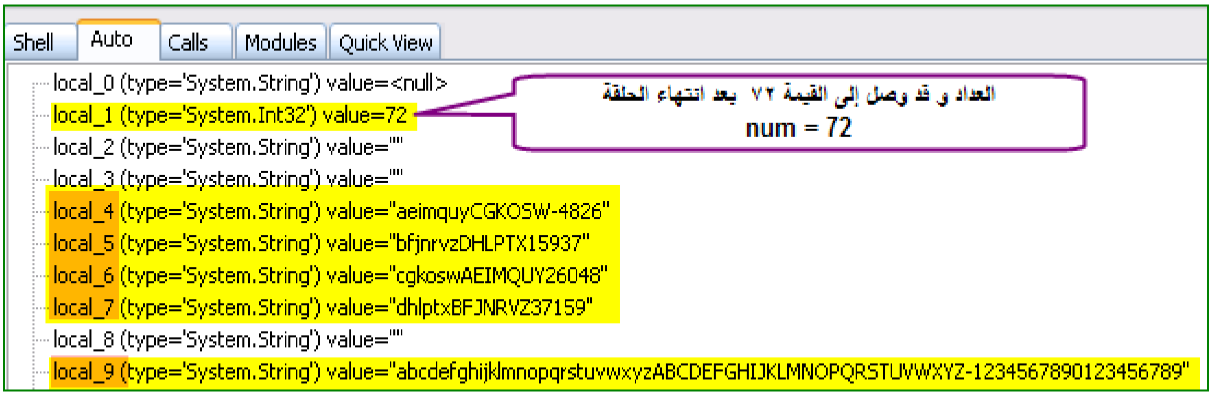
9) اضغط Register وسيتوقف المنقح عند نقطة التوقف التي وضعناها سابقاً.

نضع نقطة توقف ثانية عند السطر : L_00C0 (وضعناها هنا؛ لأنها السطر التالي لانتهاء الحلقة) .



نضغط المفتاح F7 أو الزر وسيتوقف المنقح عند نقطة التوقف L_00C0 .

ننظر إلى اللوحة Auto ونلاحظ القيم الأربع للسلاسل النصية (Str4, Str5, Str6, Str7) .



Str4 ="aeimquyCGKOSW-4826"

Str5 ="bfjnrvzDHLPTX15937"

Str6 ="cgkoswAEIMQUY26048"

Str7 ="dhlptxBFJNRVZ37159"

الآن عرفنا لنتج الحلقة السابقة وما هي المتغيرات الأربع الناتجة عنها؟ لذا سننتقل إلى السطر التالي :

```
str8 = (str4 & str5 & str6 & str7)
```

الآن البرنامج يقوم بجمع السلاسل الأربع السابقة في متغير مستقل Str8 إذن ستكون قيمته هي :

Str8 = "aeimquyCGKOSW-4826bfjnrvzDHLPTX15937cgkoswAEIMQUY26048 dhlptxBFJNRVZ37159"

بعدها يكتفي البرنامج بأخذ أول 9 محارف من كل سلسلة من السلاسل الأربع :

```
str4 = Strings.Mid(str4, 1, 9)
str5 = Strings.Mid(str5, 1, 9)
str6 = Strings.Mid(str6, 1, 9)
str7 = Strings.Mid(str7, 1, 9)
```

فتصبح السلاسل الأربع كالتالي، وهي التي سنعتمدها في بقية حديثنا :

Str4 = aeimquyCG
Str5 = bfjnrvzDH
Str6 = cgkoswAEI

Str7 = dhlptxBFJ

رغم تعقيد الحلقة التالية إلا أن مبدأها بسيط جداً :

```
Dim num4 As Integer = (StrIn.Length - 1)
Dim i As Integer = 0
Do While (i <= num4)
  If (Strings.InStr(str4, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary) <> 0) Then
    str2 = (str2 & Strings.InStr(str4, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary).ToString & "1")
  ElseIf (Strings.InStr(str5, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary) <> 0) Then
    str2 = (str2 & Strings.InStr(str5, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary).ToString & "2")
  ElseIf (Strings.InStr(str6, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary) <> 0) Then
    str2 = (str2 & Strings.InStr(str6, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary).ToString & "3")
  ElseIf (Strings.InStr(str7, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary) <> 0) Then
    str2 = (str2 & Strings.InStr(str7, Conversions.ToString(StrIn.Chars(i)), CompareMethod.Binary).ToString & "4")
  End If
  i += 1
Loop
```

– طول الحلقة يساوي طول السلسلة المررة لها (تذكر أن البرنامج يمرر أولاً أول 9 أحرف من السريال المدخل).

– عدد مرات تكرار الحلقة (9) مرات، ويتم في كل مرة أخذ حرف من الأحرف التسعة.

– الدالة InStr : تبحث عن الحرف، هل هو موجود ضمن السلسلة النصية ؟

إذا كان موجوداً : تعيد موقعه ضمن السلسلة؛ مثلاً: "InStr (Str4, "a) ستبحث عن الحرف a في السلسلة Str4 ولاحظ أن موقعه = 1 إذن InStr(Str4,"a") = 1.

إذا لم يكن موجوداً : تعيد القيمة 0

–– فالبرنامج يتأكد :

- هل الحرف موجود في السلسلة Str4 ::: إذن : أضف موقعه + الرقم 1
- هل الحرف موجود في السلسلة Str5 ::: إذن: أضف موقعه + الرقم 2
- هل الحرف موجود في السلسلة Str6 ::: إذن: أضف موقعه + الرقم 3
- هل الحرف موجود في السلسلة Str7 ::: إذن: أضف موقعه + الرقم 4

–– ولكن لاحظ : أن البرنامج في الحدث BTRClick يقارن ناتج أول 9 أحرف مع الرقم (112244617383124392).

```
[text] = Me.Ts.Text
If (([text].Length > 16) AndAlso (RHA.GetValue(Strings.Mid([text], 1, 9)) = "112244617383124392")) Then
```

و الحلقة السابقة تعطي ثنائيات بالصيغة التالية :

موقع المحرف في السلسلة + (1 أو 2 أو 3 أو 4 … حسب السلسلة التي تحتويه).

لنحلل الرقم (112244617383124392) :

Str4 = aeimquyCG
Str5 = bfjnrvzDH
Str6 = cgkoswAEI
Str7 = dhlptxBFJ

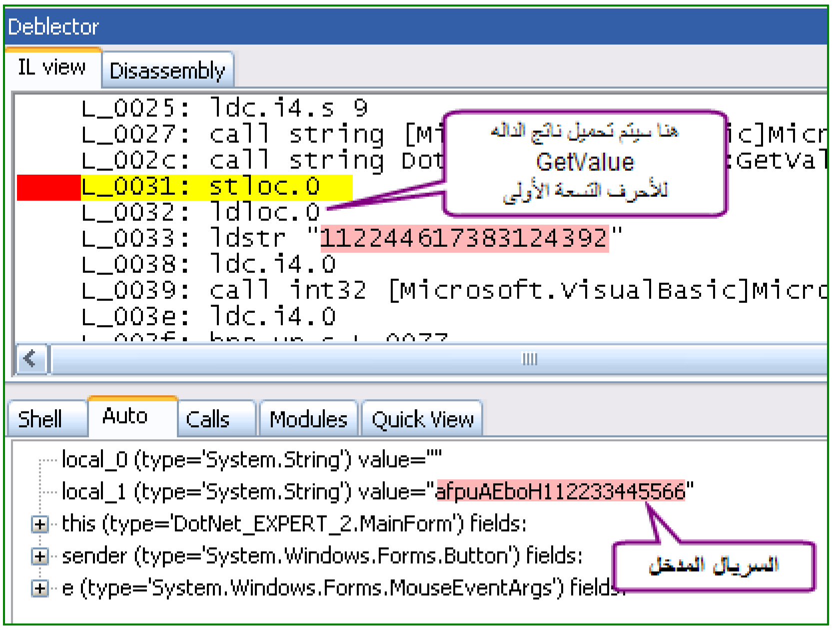
1	1 Str4	2	2 Str5	4	4 Str7	6	1 Str4	7	3 Str6	8	3 Str6	1	2 Str5	4	3 Str6	9	2 Str5
a		f		p		u		A		E		b		o		H	
afpuAEboH																	

– إذن أول 9 أحرف من السريال هي : afpuAEboH

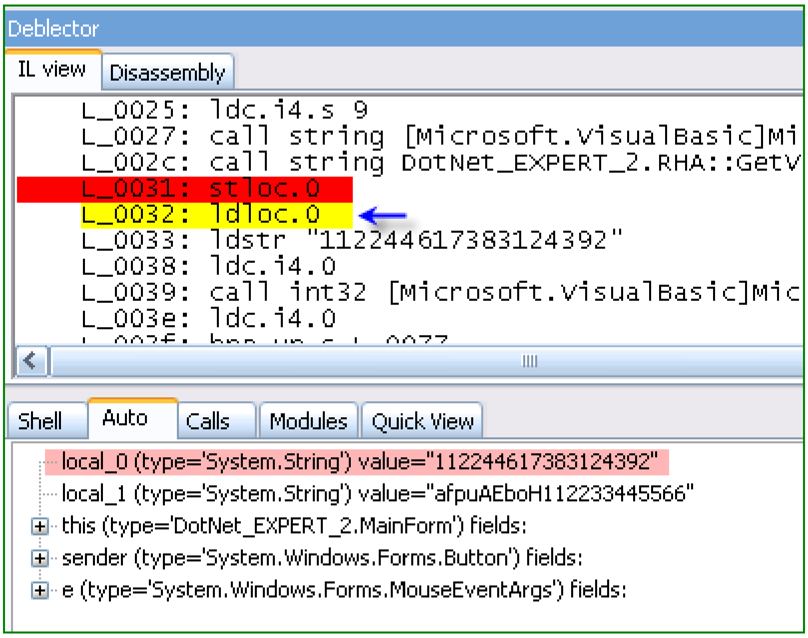
– كيف سنتأكد من أن هذه الأحرف صحيحة؟ لنشغل البرنامج ولنقم بإدخال السريال: afpuAEboH112233445566

ولنضع نقطة توقف ضمن الإجراء BTRCLICK عند السطر L_0031 :

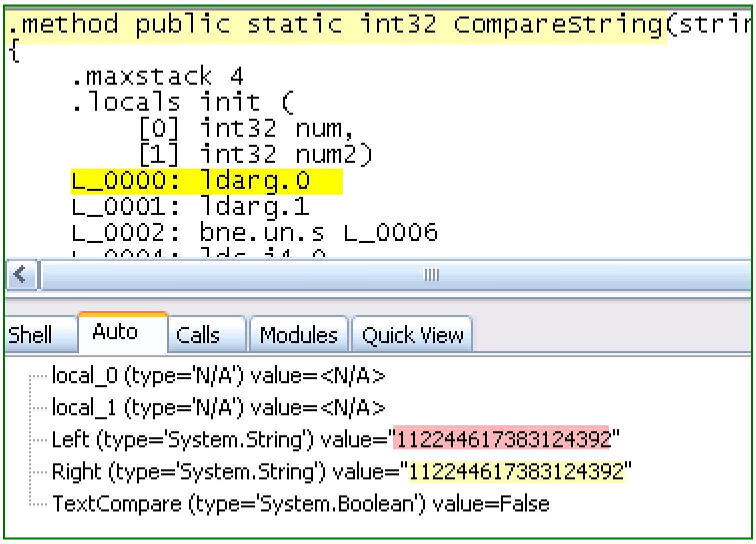
سيتوقف البرنامج عندها :



– نضغط الآن F10 مرتين ونلاحظ ظهور ناتج الدالة GetValue للأحرف التسعة الأولى :



– ومن ثم المقارنة (بواسطة F11 يمكنك الدخول إلى دالة المقارنة) ونلاحظ تطابق القيمتين، إذن تحليلنا كان صحيحاً، والأحرف التسعة من السريال هي : afpuAEboH



– نلاحظ في الإجراء BTRClick أن البرنامج يدفع الأحرف الثمانية التالية للدالة GetValue ويقارن الناتج مع 4157.

– لنحلل الرقم 4157 :

4 : أخذ المحرف الرابع

1 : من السلسلة النصية Str4

5 : أخذ المحرف الخامس

7 : من السلسلة النصية ؟ (كيف 7 ونحن لدينا إما 1 أو 2 أو 3 أو 4 وهي تدلنا على السلسلة التي نأخذ منها؟)

– حقيقة هذه هي الحركة التي سببت صعوبة الحل، لنراجع الدالة GetValue وننزل إلى الأسفل وعندها سنرى :

```
Dim str As String = str2
If Not Versioned.IsNumeric(StrIn) Then
    Return str
End If
Do While (start < 9)
    str3 = (str3 & str8.Chars(Conversions.ToInteger(Strings.Mid(StrIn, start, 2))).ToString)
    start = (start + 2)
Loop
Return str3
```

– إذن الدالة تقارن القيمة الداخلة لها، فإذا لم تكن أرقاماً Not IsNumeric فإنها تعيد ناتج الحلقة التي تحدثنا عنها منذ قليل.

– أما إذا كانت القيمة المدخلة لها أرقاماً فقط (لا يوجد حروف) فإنها تقوم بحلقة مختلفة وتعيد ناتج هذه الحلقة Str3 .

أولاً : الجزء الثاني من السريال مؤلف أرقام .

ثانياً: الحلقة أصغر من 9 (وهذا يتوافق مع كون الجزء الثاني من السريال مؤلف من 8 أرقام).

ثالثاً: يتم زيادة عداد الحلقة 2 في كل دورة (0 , 2 , 4 ,) Start = Start + 2 .

رابعاً: يتم أخذ محرفين في كل دورة Mid(StrIn,Start,2) .

- Mid(StrIn,0,2) : أول رقمين
- Mid(StrIn,2,2) : ثاني رقمين
- Mid(StrIn,4,2) : ثالث رقمين
- Mid(StrIn,6,2) : رابع رقمين

خامساً: يتم في كل دورة إيجاد المحرف من السلسلة النصية Str8 الذي موقعه قيمة الرقمين، ويضاف إلى الناتج.

ونعلم أن الناتج يجب أن يكون 4157

و السلسلة النصية Str8 :

Str8 = "aeimquyCGKOSW-4826 bfjnrvzDHLPTX15937cgkoswAEIMQUY26048 dhlptxBFJNRVZ37159"

لنحلل الرقم: إذن لاحظ ترتيب الأرقام المعلمة بالأزرق (يبدأ الترقيم بالرقم 0 وليس 1) :

- 4 : ترتيبه (14) في السلسلة Str8
- 1 : ترتيبه (31) في السلسلة Str8
- 5: ترتيبه (32) في السلسلة Str8
- 7: ترتيبه (35) في السلسلة Str8

إذن فالجزء الثاني من السريال – المؤلف من 8 أرقام والذي ينتج الرقم (4157) من خلال الدالة GetValue– هو : 14313235

فيكون السريال النهائي : afpuAEboH14313235

لنقم بإدخال هذا السريال ونضغط Register ستظهر لنا العبارة Registered في المربع النصي RegCode

هل تستطيع أن تعرف من أين أتت Registered ؟

5. خامساً: احتمالات السريالات ضمن الكيجن

استطاع الإخوة العثور على 8 سريالات كحد أقصى، وهذه هي الطريقة المنهجية لإحصاء احتمالات السريالات البالغة 16 سريالاً :

Str8 = "aeimquyCGKOSW-4826bfjnrvzDHLPTX15937cgkoswAEIMQUY26048dhlptxBFJNRVZ37159"

- الرقم الأول يجب أن يكون 4: وهو يملك الموقعين (14 – 52)
- الرقم الثاني يجب أن يكون 1: وهو يملك الموقعين (31 – 69)
- الرقم الثالث يجب أن يكون 5: وهو يملك الموقعين (32 – 70)
- الرقم الرابع يجب أن يكون 7: وهو يملك الموقعين (35 – 68)

[4] Location	[1] Location	[5] Location	[7] Location	Serials
14	31	32	35	14313235
			68	14313268
		70	35	14317035
			68	14317068

	69	32	35	14693235
			68	14693268
		70	35	14697035
			68	14697068
52	31	32	35	52313235
			68	52313268
		70	35	52317035
			68	52317068
	69	32	35	52693235
			68	52693268
		70	35	52697035
			68	52697068

يمكن كتابة كيجن للبرنامج الهدف بواسطة دلفي مستخدماً مصفوفة تحتوي على احتمالات الجزء الثاني من السريال (يستخدم عناصرها بشكل عشوائي) ويضاف لها الجزء الأول من السريال, أو يمكن اعتماد المنهج الترتيبي, ويعتبر المبدأ العشوائي هو الأكثر استخداماً في كتابة مثل هذه الكيجنات. و يمكن كتابة الكود بالشكل التالي :

```
var
  Form1: TForm1;
  I: Integer = 0;
Const
Array2S : Array[0..15] of String =('14313235'
,'14313268'
,'14317035'
,'14317068'
,'14693235'
,'14693268'
,'14697035'
,'14697068'
,'52313235'
,'52313268'
,'52317035'
,'52317068'
,'52693235'
,'52693268'
,'52697035'
,'52697068');

implementation
{$R *.dfm}
procedure TForm1.BTGenClick(Sender: TObject);
begin
//Method #1 ===> Random
Randomize;
TReg.Text := 'afpuAEboH' + Array2S[Random(15)];
//Method #2 ===> Ordinal
if I > 15 Then I := 0;
TReg.Text := 'afpuAEboH' + Array2S[I];
I:= I + 1;
end;
```

نتمنى أن تكونوا قد استمتعتم واستفدتم من قراءة هذا الدرس، كما نرجو من الله تعالى أن يكون عملنا خالصاً لوجهه الكريم .

تحليل تطبيقات الدوت نت [2]

.Net Applications Analysis (2)

I. مقدمة و ايضاحات

تحدثنا في الجزء الأول عن الإمكانيات المتقدمة التي تقدمها لنا الأداة Reflector , وذكرنا أن الإضافات المصممة لهذه الأداة تزيد من قوة هذه الأداة، ولعل أهم هذه الإضافات Deblector : و. Reflexil

أثناء عمليات الباتشينغ تشكل الإضافتين Deblector و Reflexil ثنائياً رائعاً ومتكاملاً لتحليل وتعديل تطبيقات الدوت نت.

يلعب : Deblector دور منقح يحلل الكود أثناء التنفيذ ويرسم صورة واضحة لما يمكن إنجازه من خلال الأداة. Reflexil

ويلعب : Reflexil الدور النهائي في التعديل على التطبيق وحفظ الملف المعدل بعد الفهم التام لما يجري.

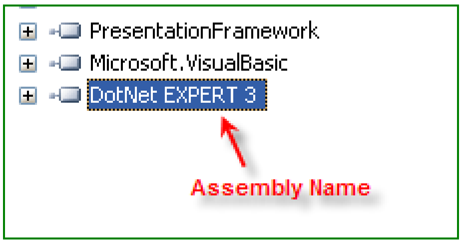
-تم تصميم التمرين DotNet Expert 3 لشرح إمكانيات الأداة Reflixel وكيفية التعامل معها.

Reflector Reflexil	

DotNet EXPERT 3.exe	

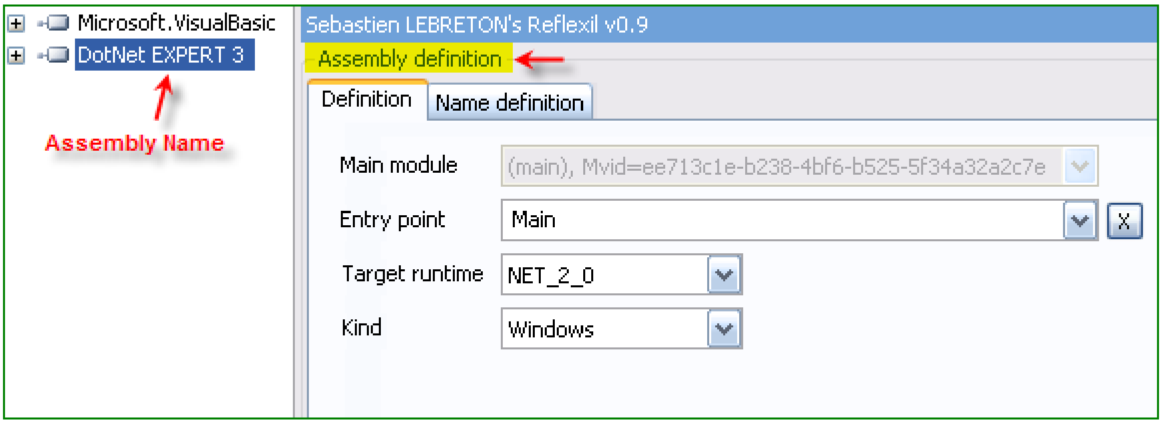
I.1 أولاً: تعديل Assembly Name وحفظ الملف المعدل

-لنسقط التمرين في Reflector وعندها سنلاحظ ظهور اسم التمرين على القائمة اليسرى، هذا الاسم يعرف بـــ Assembly Name :

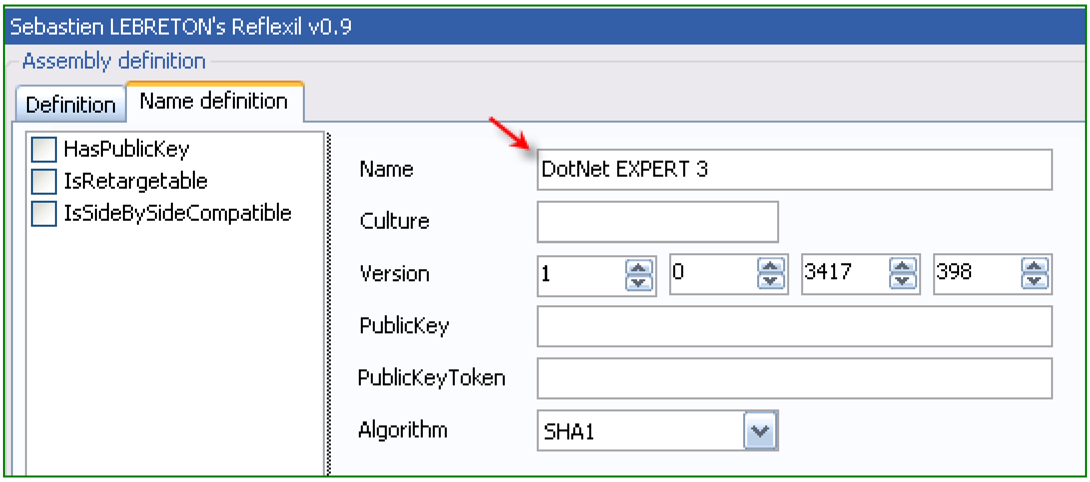


و كوننا سنقوم بعدد من التعديلات فإذا ظهر لنا التمرين بنفس الاسم دوماً فسيتم استبدال الملف الجديد المعدل بالملف القديم وسنفقد إمكانية المقارنة مع التمرين الأصلي و سنفقد متابعة التغييرات التي نجريها.

لنفتح الآن الأداة Reflexil وإذا كان الـــ Assembly Name معلم فعندها ستكون النافذة هي Assembly Definition



لننتقل إلى الصفحة Name definition ونغير الـــ : Assembly Name

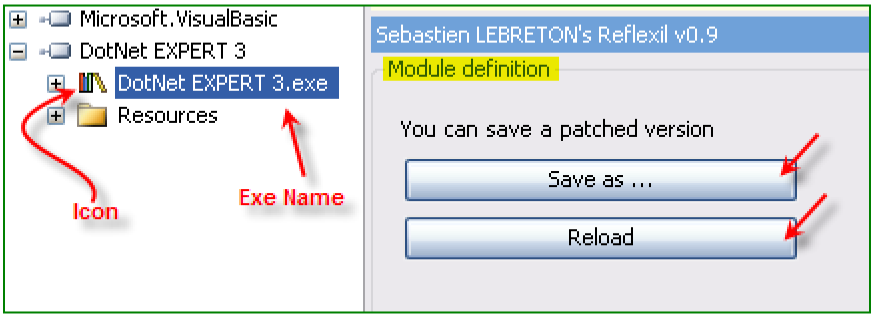


سنغيره إلى Patched_1 لتكون الملفات التالية مرتبة Patched_2 ... Patched_3... وهذا يفيدنا في متابعة التغييرات:



لنقوم الآن بحفظ التعديل في ملف جديد.

نوسع القائمة (+) المجاورة لـــ Assembly Name وسنلاحظ ظهور Exe Name وبجواره أيقونة عبارة عن كتب، وعند الضغط على اسم الملف التنفيذي تغير نافذة الإضافة Reflexil إلى النافذة: Module definition

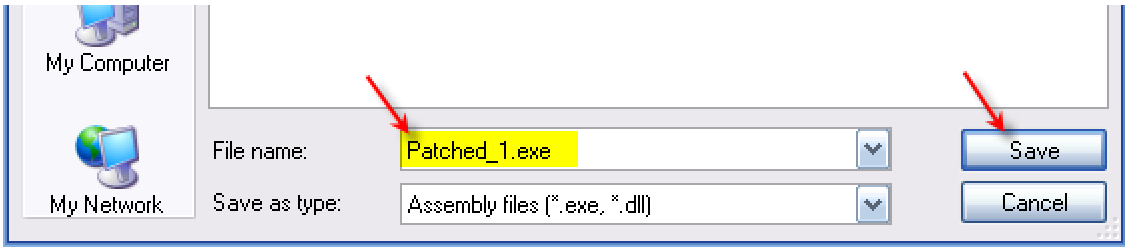


كما تلاحظ ظهر لدينا زرين هما :

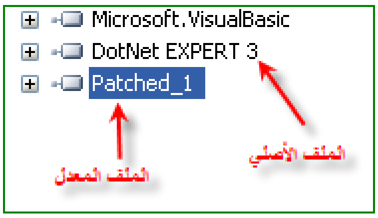
Save as : يسمح لنا بحفظ النسخة المعدلة من البرنامج.

Reload: يسمح لنا بإعادة تحميل الملف الأصلي ومسح جميع التغيرات التي أجريناها.

لنضغط Save as ونحفظ الملف باسم: Patched_1

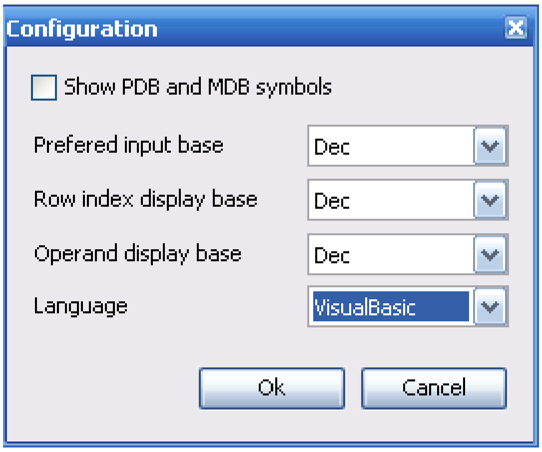


بعد حفظ الملف نتوجه إلى مكان الحفظ , نسقط الملف Patched_1.exe في Reflector ونلاحظ ظهور اسم جديد Patched_1 :



1.2 ثانياً: إعدادات الأداة Reflexil

نلاحظ أسفل نوافذ Reflexil ارتباطاً نصياً يحمل الاسم [Configure Reflexil ..] وعند الضغط عليه تفتح نافذة جديدة وسنحاول أن نختار الحقول كما في الصورة التالية، ويمكنك التعديل بها بما يناسبك:



1.3 ثالثاً : النافذة Method definition

تعتبر هذه النافذة القلب النابض للأداة Reflexil , ومركز العمليات أثناء التعديل على تطبيقات الدوت نت.

تتألف هذه النافذة من صفحات جزئية هامة وتظهر عند استعراضك لأحد الإجرائيات في البرنامج Reflector وهي:

1.4 الصفحة: Instructions

تقوم بعرض رقم سطر الكود + الموقع (Offset) + تعليمات اللغة الوسيطة الموافقة (IL) + عوامل التشغيل التابعة لهذا السطر.

1.5 الصفحة: Variables

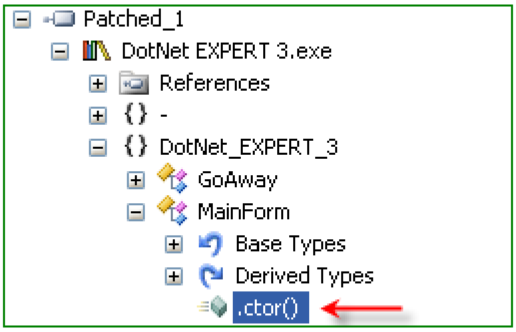
تعرض لك أسماء المتغيرات وأنواعها، ولكن بأسماء ترتيبية V_0, V_1, وذلك حسب تسلسل ورودها ضمن نص الكود الموافق.

2. تمرين تطبيقي لاستغلال الصفحتين: Instructions & Variables

سنتناول هنا تخطي الإجراء BRVClick وهو يمثل الضغط على الزر Register Version.

بداية كيف عرفت أن الحدث BRVClick يمثل الضغط على الزر Register Version ؟

إذا نظرنا إلى العناصر الخاصة بـــ MainForm نجد في الأعلى ctor() وهو اختصار لـــ Constructor أي عمليات بناء النموذج.

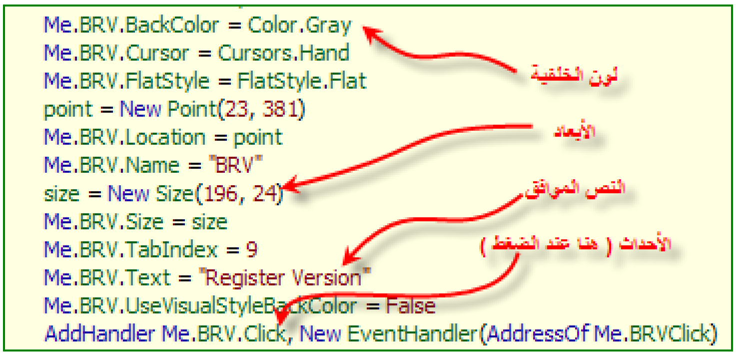


عند الدخول إليه سنلاحظ تعريفات مع قيم بدائية للمتحولات العامة المستخدمة في النموذج, وإجراء أعداد المكونات الخاصة بالنموذج Me.InitializeComponent

```
Public Sub New()
   Me.T1 = 0
   Me.IntT = 0
   Me.StrArr = New String() { "Failure", "Success", "Failure", "Maybe Incorrect Input...
   Me.InitializeComponent
End Sub
```

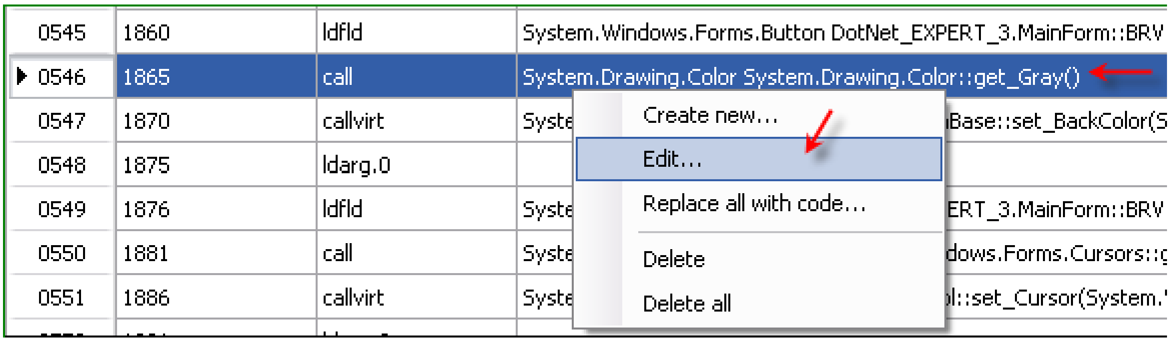
عند الدخول لهذا الإجراء سنلاحظ تعريف المكونات من حيث النوع (Button, PicutreBox, RadioButton ...) والتسمية وغيرها.

ونلاحظ هنا الإعدادات الخاصة بالزر: Register Version

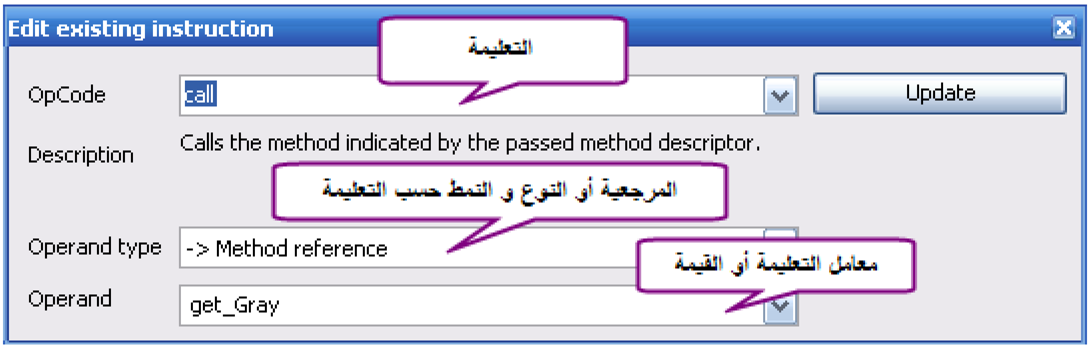


من باب التسلية : دعونا نقوم بتغيير لون الخلفية إلى الأخضر والكتابة إلى: My First Button

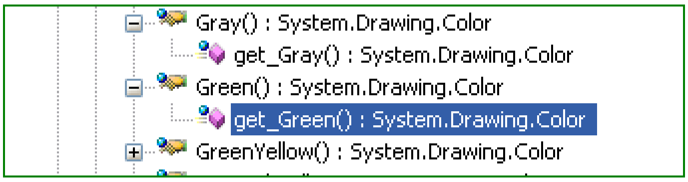
في اللوحة Instruction نختار السطر 546 الذي يحتوي على الدالة GetGray ونضغط بزر الفأرة الأيمن ونختار: Edit



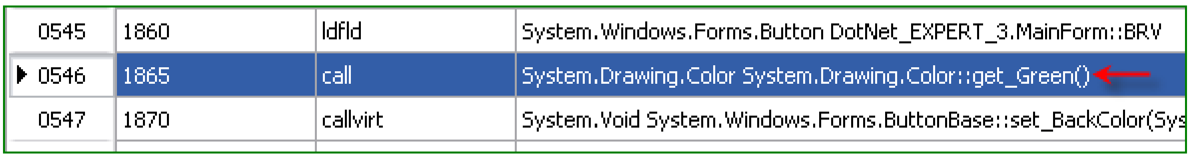
ستخرج لنا نافذة التحرير:



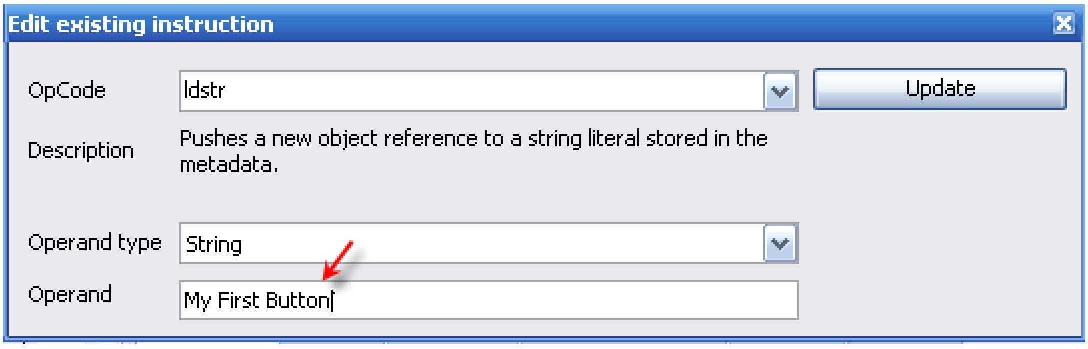
نضغط على Get_Gray لتخرج نافذة جديدة نختار منها (Get_Green السطر التالي:)



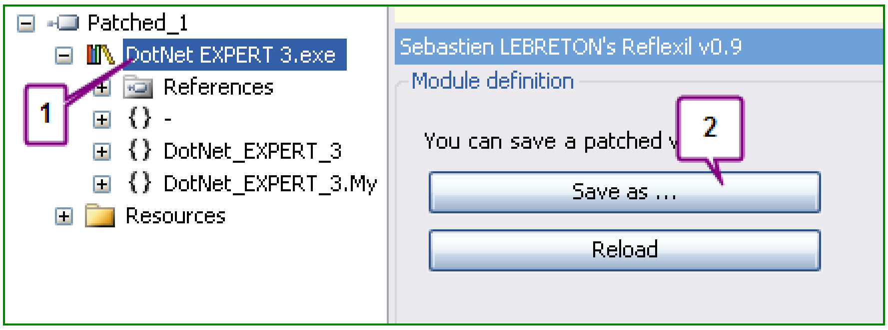
ثم نضغط على الزر Update لحفظ التغييرات، فتغلق النافذة، ونلاحظ أن التعديل قد تم على السطر 546:



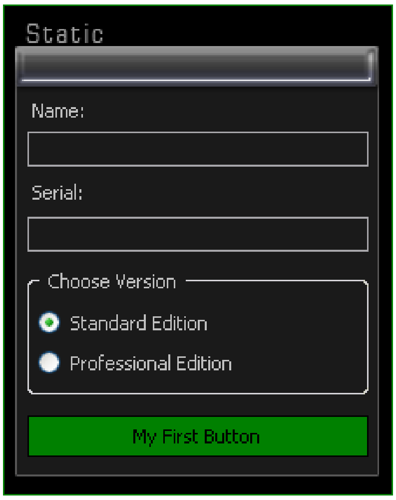
نكرر العلمية من أجل السطر 586 والذي يحتوي على العبارة Register Vesion ولذلك هنا نكتب بدل النص My First Button :



الآن نحفظ الملف بالضغط على Exe Name ثم نختار Save as ونسمي الملف: For Fun



نشغل الملف الناتج (For Fun.exe) ونشاهد التغييرات:



-نعود إلى موضوعنا ونخطي الحدث: BRVClick

بداية نختار Exe name أسفل الموديل Patched_1 : ونضغط من Reflixel الزر Reload لإزالة التغيرات التي أجريناها منذ قليل.

نتصفح الآن الكود الموافق ونحاول فهم ما يجري لإجراء التعديلات المطلوبة بدقة:

```
Me.IntT = 0
If ((Me.TNS.Text = "") Or (Me.TSS.Text.Length < 5)) Then
    Me.MacroMe(2, 3, 0)
Else
```

في البداية : يتأكد البرنامج من أن الاسم ليس فارغاً وأن طول السريال ليس أقل من 5 أحرف.

في حال عدم تحقق الشرط السابق سينفذ الإجراء MacroMe ويمرر له المعاملات (البرميترات Parameters) 2 – 3 – 0 بالترتيب.

لندخل إلى الإجراء MacroMe ونرى ماذا يفعل؟

```
Public Sub MacroMe(ByVal StrUp As Integer, ByVal StrDn As Integer, ByVal IntIn As Integer)
    StrUp = (StrUp * Me.IntT)
    IntIn = (IntIn * Me.IntT)
    Me.LTT.Text = Me.StrArr(StrUp)
    Me.TRES.Text = Me.StrArr(StrDn)
    Me.timer1.Enabled = True
    Me.PB.BackgroundImage = Me.Img
End Sub
```

ضرب القيمتين الأولى و الثالثة بالمتغير العام IntT

اظهار النص الموافق من المصفوفة StrArr

طبعاً إذا كنت تذكر فالمصفوفة StrArr موجودة عند ctor وعناصرها بالترتيب:

```
    Me.IntT = 0
    Me.StrArr = New String() { "Failure", "Success", "Failure", "Maybe Incorrect Input...
    Me.InitializeComponent
End Sub
```

رقم العنصر	النص
0	Failure
1	Success
2	Failure
3	Maybe Incorrect Input
4	Standard Edition
5	Professional Edition
6	What is Upper Text .. This Is Result ..

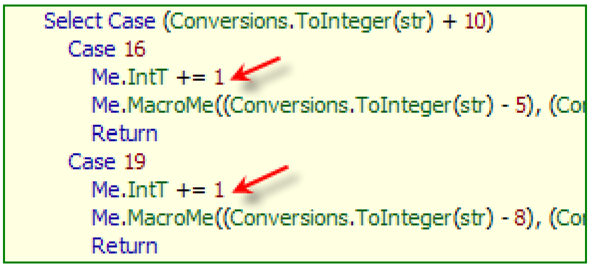
معنى الإجراء : MacroMe(2, 3, 0) أظهر النص – (2) Failure والنص – (3) Maybe والأيقونة (0) الخطأ.

ولكن انتبه ضرب الأول والثالث بقيمة المتحول IntT : وكما لاحظت أن قيمته البدائية هي (0.)

و بالتالي تتحول القيمة إلى MacroMe (2,3,0) : إلى. MacroMe (0,3,0)

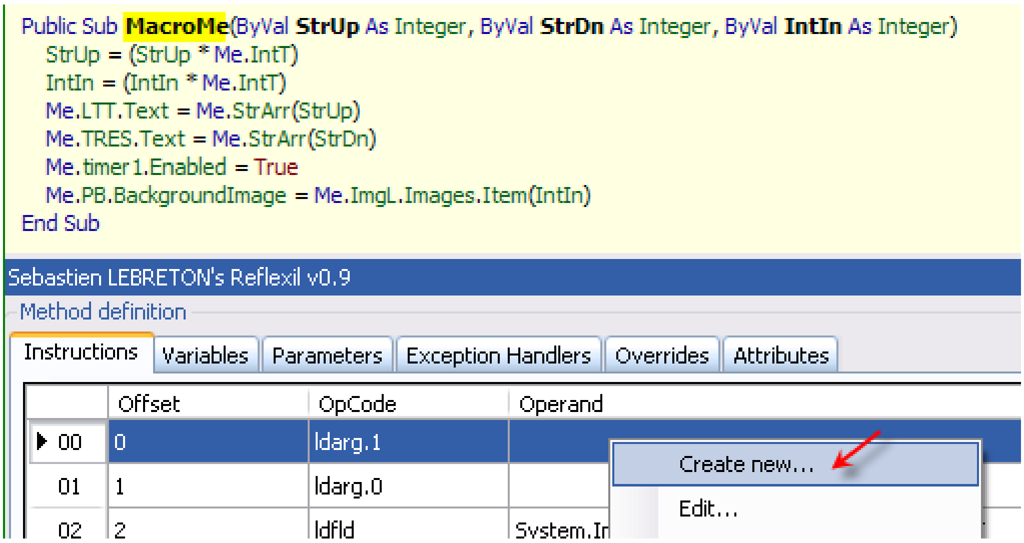
ولكن إذا كانت قيمة هذا المتحول (1) : فستبقى القيم كما مررت للإجراء . كان الهدف من هذه العملية ظهور نتيجة الخطأ دوماً في حال تم التلاعب بقفزات (الجمل الشرطية) للبرنامج.

-لاحظ في بقية الإجراء (BRVClick) كيف يتم إعطاء المتحول IntT القيمة 1 عند التسجيل الصحيح.

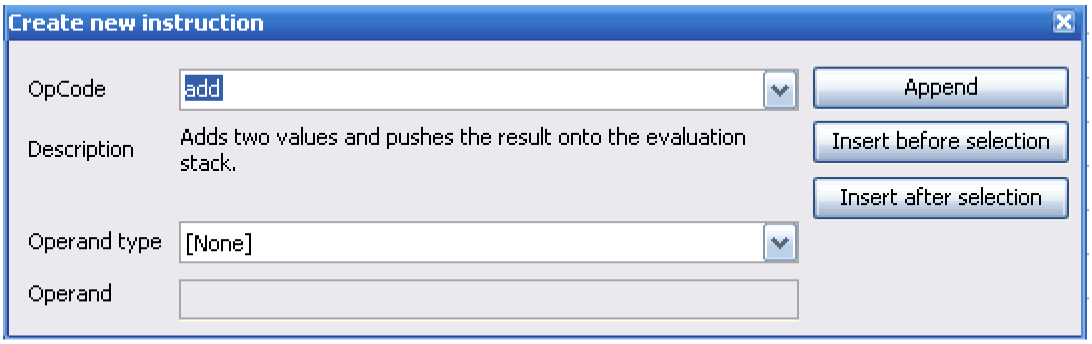


-لنحل جميع إشكاليات المتحول IntT نحتاج فقط إلى إعطاء هذا المتحول القيمة 1 عند بداية الإجراء MacroMe لنقم بذلك.

نفتح الإجراء MacroMe ونتوجه إلى اللوحة Instructions ونؤشر على السطر الأول ونختار الأمر Create New من القائمة المنبثقة بالضغط بالزر الأيمن:



ستخرج لنا نافذة جديدة (مشابهة لنافذة التحرير) :



الآن علينا إضافة التعليمات اللازمة لإعطاء المتحول IntT القيمة 1 ولكن نلاحظ أن المتحول IntT هو متحول عام إذن:

إضافة مرجعية للمتحول (Me) من خلال التعليمة.ldarg.0

دفع القيمة (1) إلى قمة المكدس من خلال التعليمة.ldc.i4.1

إسناد هذه القيمة للمتحول IntT من خلال التعليمة .stfld

التعليمة الأولى:

الشرح	Operand	Operand Type	OpCode
تحميل المرجعية 0	None	None	Ldarg.0

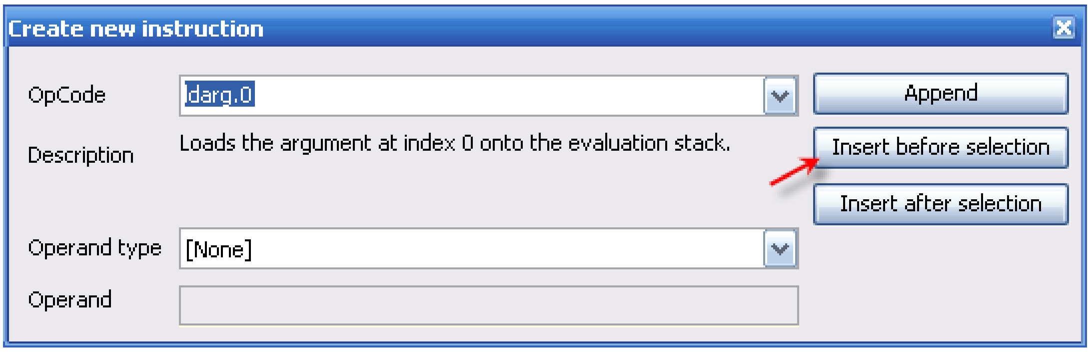
التعليمة الثانية:

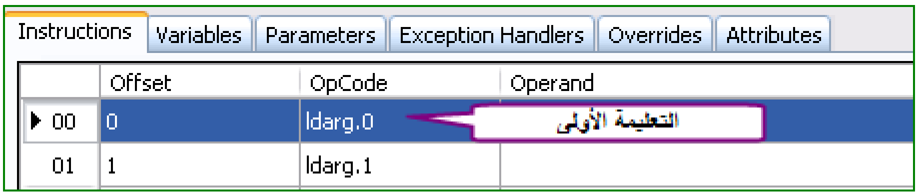
الشرح	Operand	Operand Type	OpCode
دفع القيمة 1 إلى قمة المكدس	None	None	Ldc.i4.1

التعليمة الثالثة:

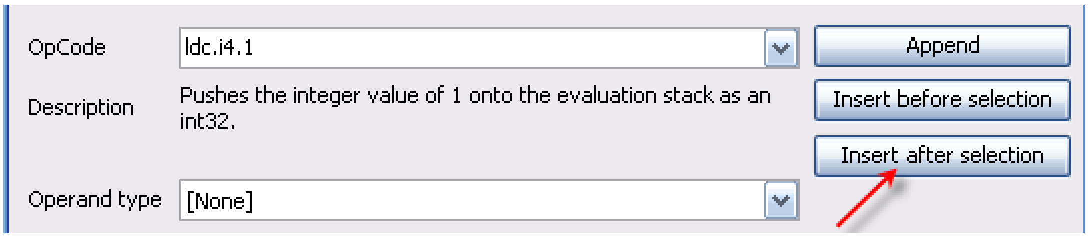
الشرح	Operand	Operand Type	OpCode
إسناد القيمة في المكدس للمتحولIntT	intT	Field reference	Stfld

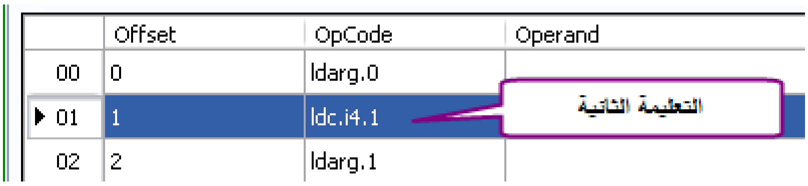
لتركيب التعليمة الأولى : فقط نكتب التعليمةldarg.0 و نضغط الزر: Insert before Selection



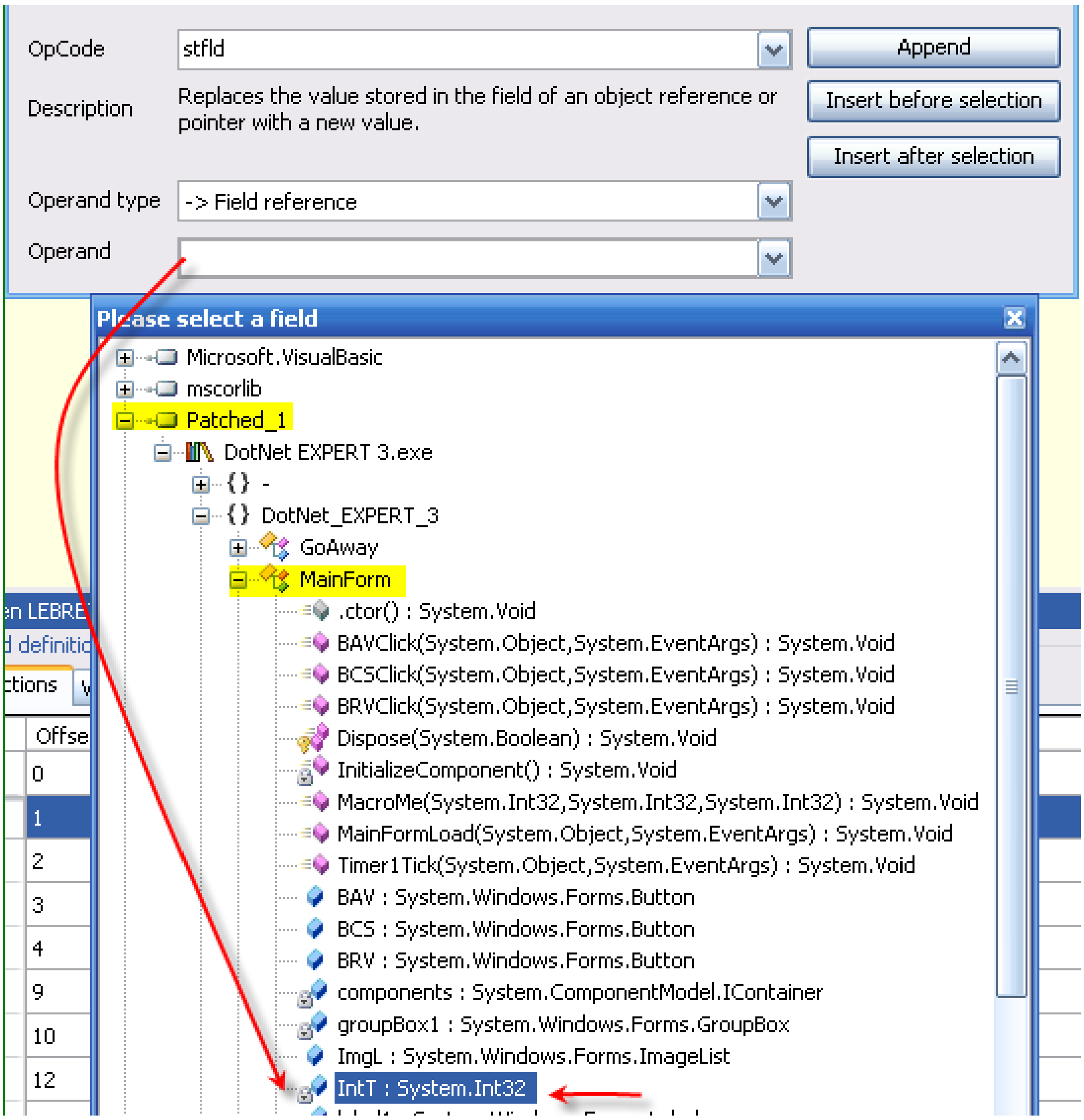


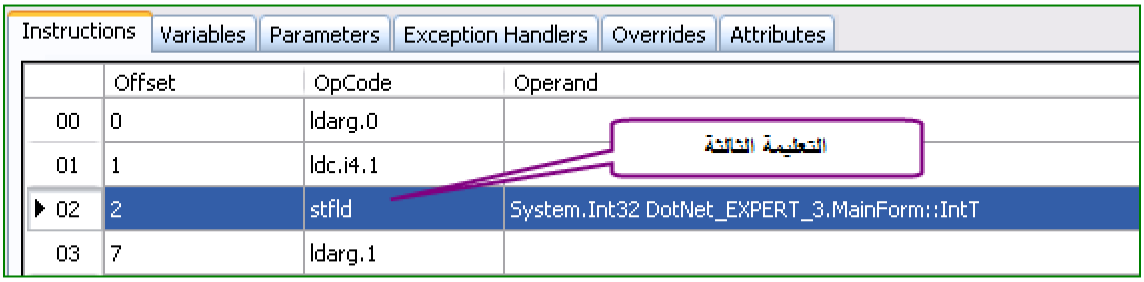
لتركيب التعليمة الثانية : نختار التعليمة الأولى ثم Create New ونكتب التعليمة ldc.i4.1 ونضغط الزر Insert before Selection :





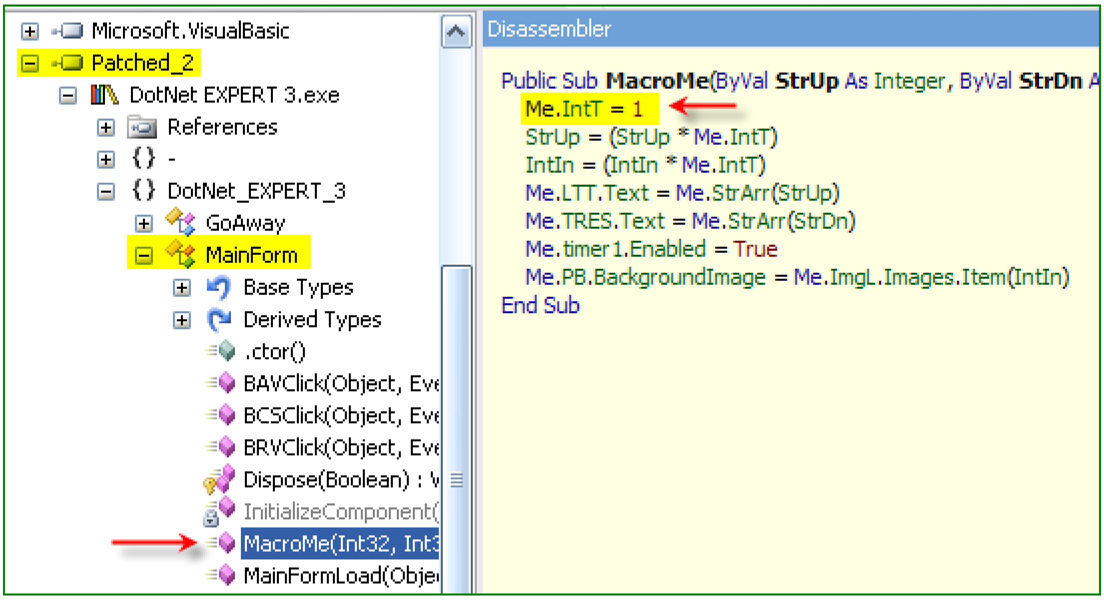
لتركيب التعليمة الثالثة : نختار التعليمة الثانية create new ويتم اختيار الحقول كما في الصورة ثم: Insert after Selection





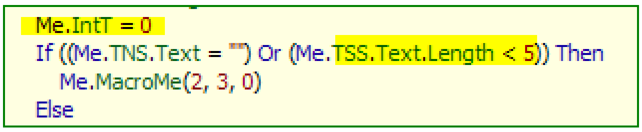
الآن نغير الـــ Assembly Name إلى Patched_2 ونحفظ الملف بالاسم Patched_2.

نحمل الملف الجديد في البرنامج Reflector ونتوجه إلى الإجراء MacroMe لنلاحظ التغيرات التي أجريناها:



نلاحظ أنه تم إسناد القيمة 1 للمتحول IntT وبالتالي تخلصنا من إشكاليات هذا المتحول نهائياً.

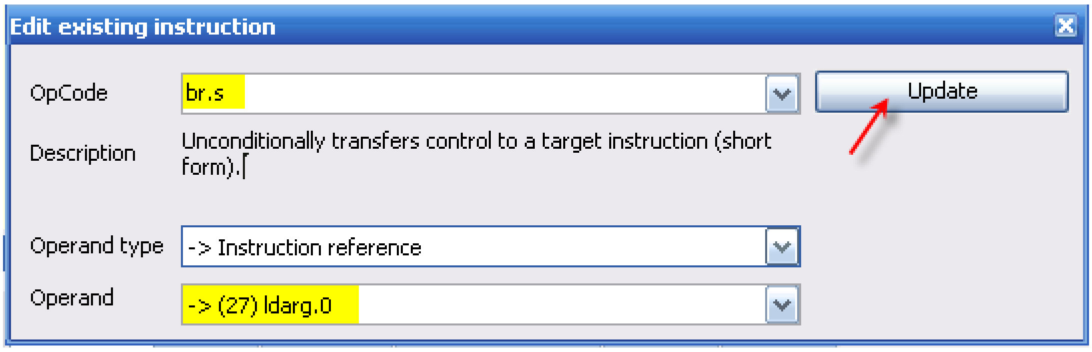
نعود مرة أخرى للإجراء (BRVClick) ونحاول تخطي محدودية الاسم الفارغ، وطول السريال أقل من 5 حروف:



من الخطأ هنا أن نعكس القفزة (لأنه في هذه الحالة ستظهر رسالة الخطأ إذا لم يكن الاسم فارغاً والسريال أطول من 5 حروف) لذلك سنوجه القفزة إلى السطر التالي وكأن الشرط غير موجود من خلال:

تغيير التعليمة brfalse.s عند السطر 20 إلى التعليمة br.s ويتم توجيه القفز إلى السطر 27 التالي لتعليمة: ret

نختار السطر 20 ثم Edit ونحرر الخصائص كما في الصورة التالية:



نشاهد في الصورة أن البرنامج يمرر الحروف الخمسة من السريال إلى الدالة: FirstServ

```
Else
  str = Services.FirstServ(Strings.Mid(Me.TSS.Text, 1, 5))
```

يتحقق البرنامج هل زر الراديو Standard Edition هو المختار؟

يتحقق من أول حرفين الناتجين عن الدالة: FirstServ

إذا كانت قيمتهم 31 سيمرر الحرفين إلى الدالة SecondServ ويتابع.

أما إذا قيمتهم 41 فسيظهر رسالة الخطأ ويخرج من الإجراء:

```
If Me.RBS.Checked Then
  Select Case Strings.Mid(str, 1, 2)
    Case "31"
      str = Services.SecondServ(Strings.Mid(str, 1, 2))
      Exit Select
    Case "41"
      str = Services.ThirdServ(Strings.Mid(str, 1, 2))
      Me.MacroMe(2, 3, 0)
      Return
  End Select
Else
```

كلمة ختامية:

نتمنى أن تكونوا قد استمتعتم واستفدتم من قراءة هذا الدرس، ونرجو من الله أن يكون عملنا خالصاً لوجهه الكريم.

كما ذكرنا آنفاً: لن نعيد أي كلمة مما ذكر في هذا الدرس, وسيقتصر حديثنا في الدروس المقبلة على رؤوس الأقلام فقط؛ لذا نرجو قراءة هذا الدرس بتمعنٍ وتدبر.

تحليل تطبيقات الدوت نت [ج]

.Net Applications Analysis (3)

الفهرس:

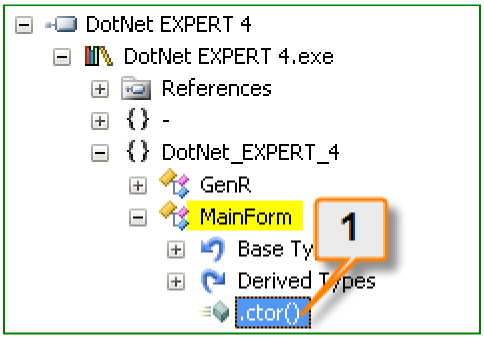
- Reflector
- Reflexil
- Deblector

DotNet EXPERT 4.exe

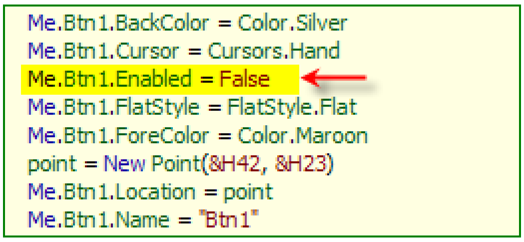
– إذا لم تقرأ الجزئين السابقين فننصحك بقراءتهما قبل البدء بقراءة هذا الدرس كي تستوعب الخطوات بشكل جيد.

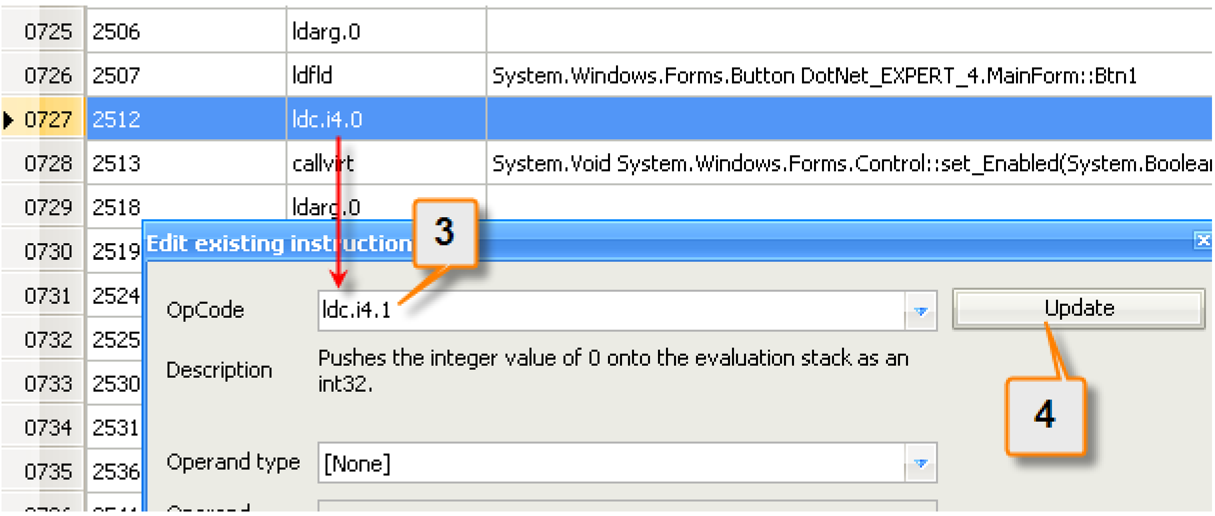
– عندما نقوم بحفظ التعديلات في ملف جديد يتم في الخطوة التالية تحميل الملف الجديد المعدل الذي حفظته للتو.

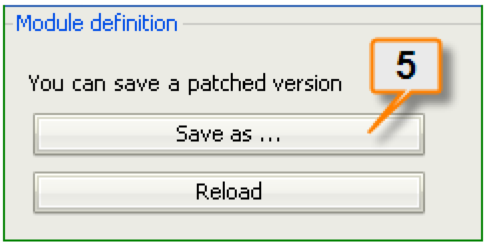
I. أولاً: تفعيل الزر Enable Me





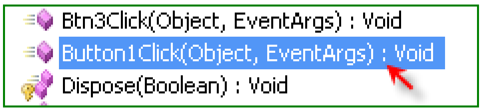






2. ثانياً: تجاوز حدث الزر Enable Me

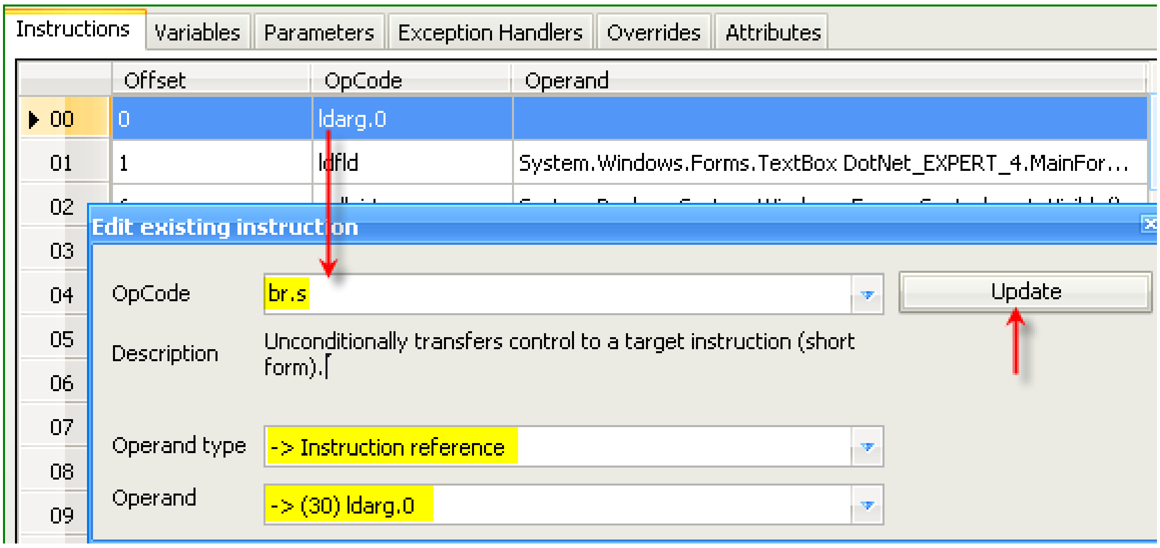
هذا الزر مسؤول عن إظهار المربع النصي والزر الخاص بالمرحلة Step2 , وسنجعله يظهر العنصرين في كل الحالات، ونتجاوز جميع الحالات الشرطية. أما فيما يتعلق بالمتحول MasterID فسنعالجه في الحدث Timer لاحقاً. تابع الخطوات :



الآن سنقفز مباشرة إلى الكود الخاص بإظهار العنصرين :

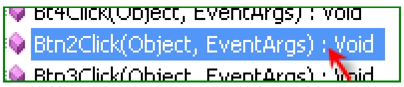
```
Me.Btn2.Visible = True
Me.Txt2.Visible = True
Me.MasterID = 1
Me.timer1.Enabled = True
```

سنستخدم تعليمة القفز غير الشرطية br.s للتوجه إلى السطر 30 :





3. ثالثاً: تجاوز المرحلة Step 2

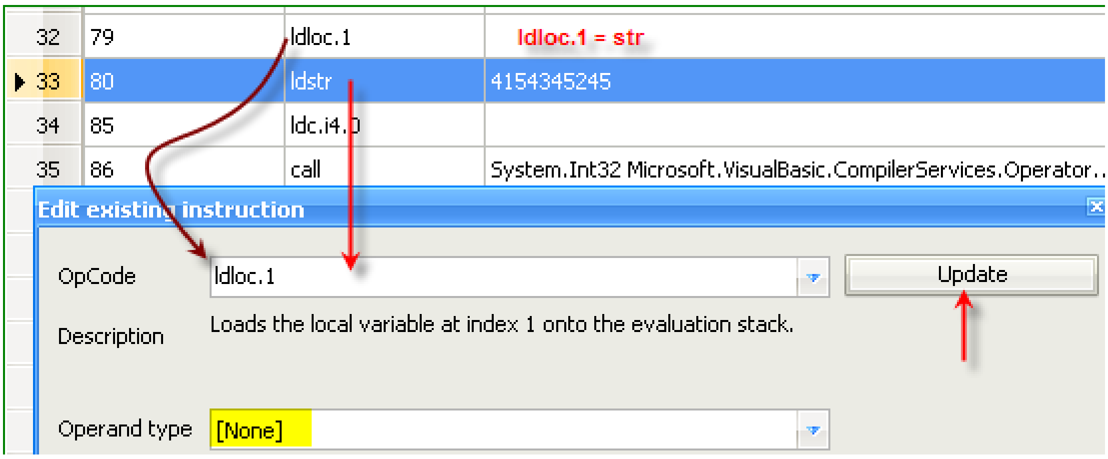


عند الدخول للإجراء السابق نلاحظ مقارنة بين ناتج تحويل قيمتنا المدخلة إلى Hex مع القيمة(4154345245 = AT4RE) :

```
If (str = "4154345245") Then
    Me.Txt3.Visible = True
    Me.MasterID += 1
    Me.timer1.Enabled = True
End If
```

يمكن هنا استخدام تعليمة قفز غير شرطية br.s إلى السطر 38 ولكننا سنستخدم طريقة أخرى؛ حيث سنضع المتحول Str بدل القيمة 4154345245 ، فتصبح المقارنة : If Str = Str Then وهي محققة على الدوام.

لاحظ المتغير loc.1 يمثل القيمة Str ويتم استدعاءها بالأمر Ldloc.1 :

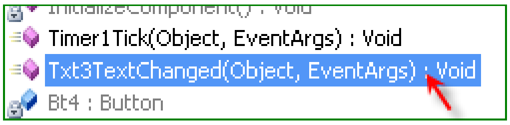




نحمل الملف الذي حفظناها ونلاحظ التغيير :

```
If (str = str) Then
    Me.Txt3.Visible = True
    Me.MasterID += 1
    Me.timer1.Enabled = True
End If
```

9. رابعاً: تجاوز المرحلة Step 3



هنا المقارنة تهدف إلى إظهار العبارة Registered Version فقط.

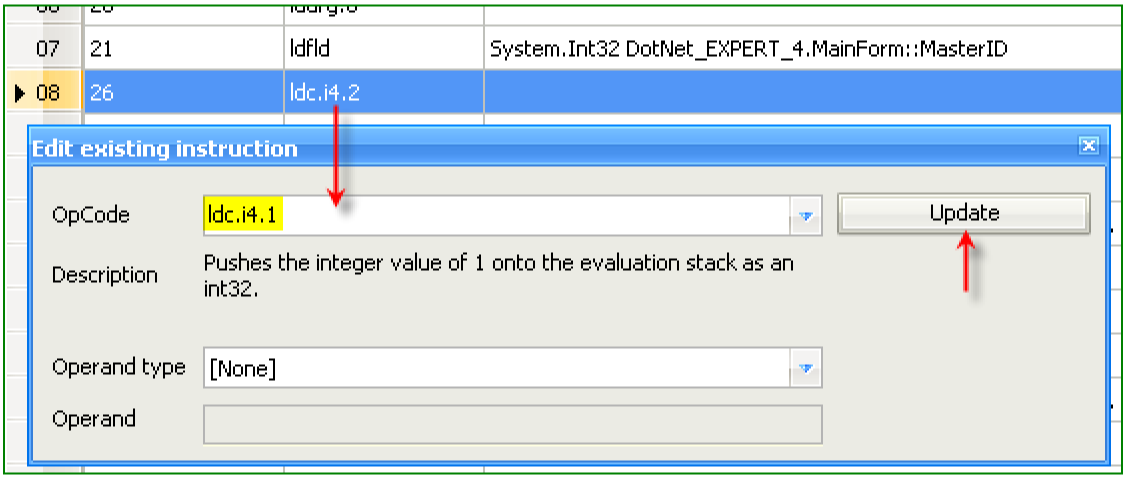
```
Public Sub Txt3TextChanged(ByVal sender As Object, ByVal e As EventArgs)
   If ((Strings.Len(Me.Txt3.Text) > 10) AndAlso (Me.MasterID = 2)) Then
      Dim str As String = (Strings.Mid(Me.Txt3.Text, 3, 1) & Strings.Mid(Me.Txt3.Text, 7, 1))
      Try
         If (Conversions.ToInteger(str) = 15) Then
            Me.Lb4.Text = "Registered Version"
            Me.MasterID += 1
            Me.timer1.Enabled = True
         End If
```

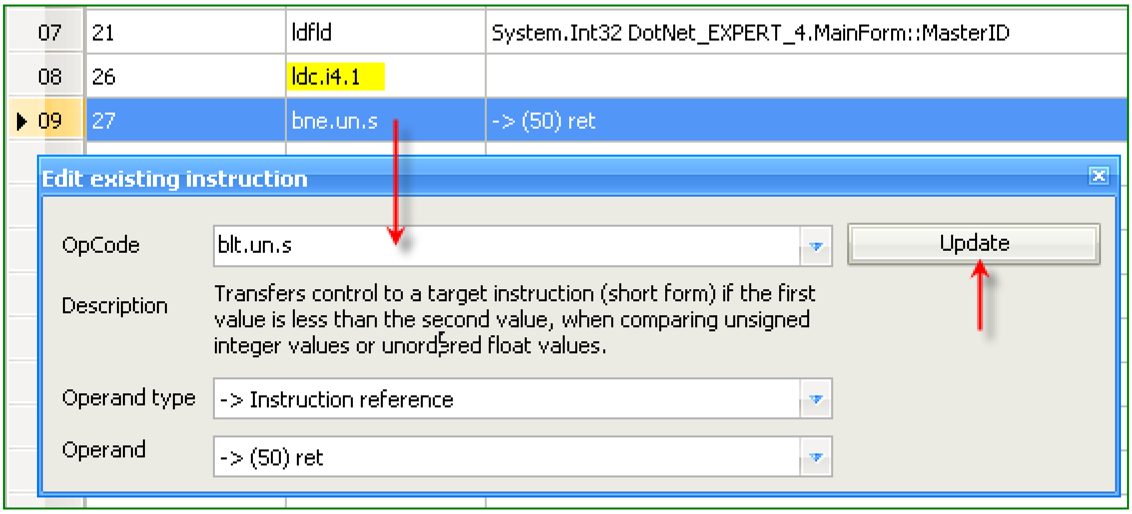
كما تلاحظ : يجب أن يكون النص المدخل اكبر من 10 أرقام (11) والمتحول MasterID يساوي 2 ، وعندها يتم أخذ المحرف ذو الترتيب (3) ويجب أن يساوي (1)، و المحرف ذو الترتيب (7) ويجب أن يساوي (5) ، إذا وضعناهم جانب بعضهم ينتج 15).

الحل :

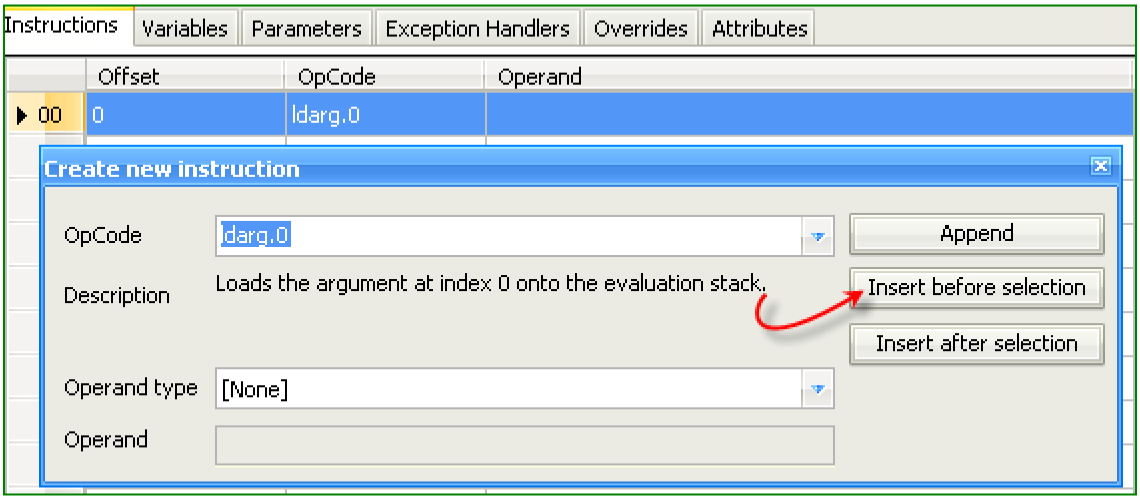
سنضع القيمة (1) بدل (2) … وسنغير المساواة إلى (أكبر من أو يساوي).

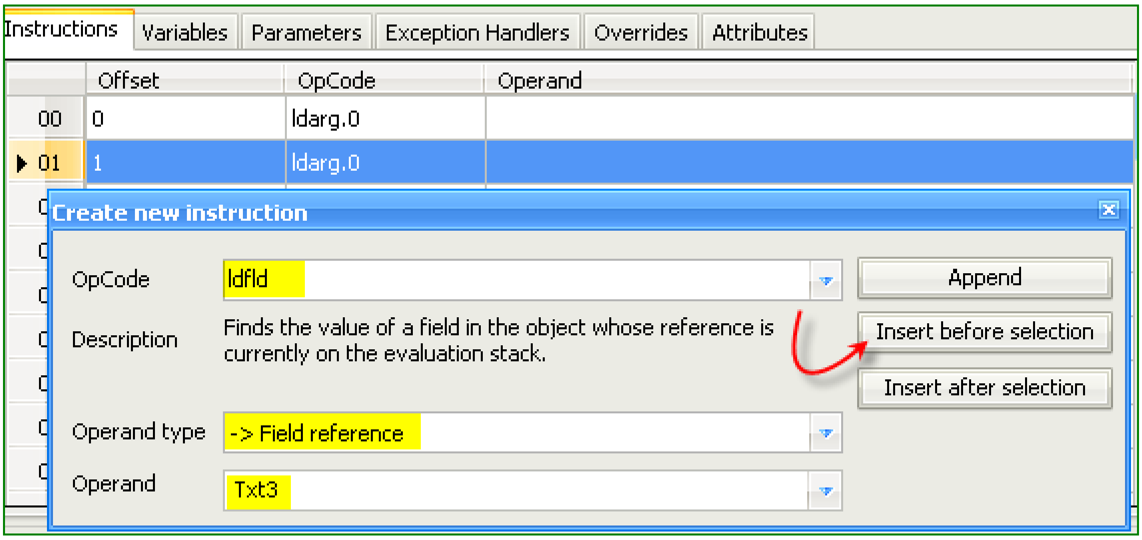
سنسند قيمة للمربع النصي Txt3 تمثل سريال صحيح وهي [00100050000] (الطول = 11 – الثالث = 1 – السابع = 5):

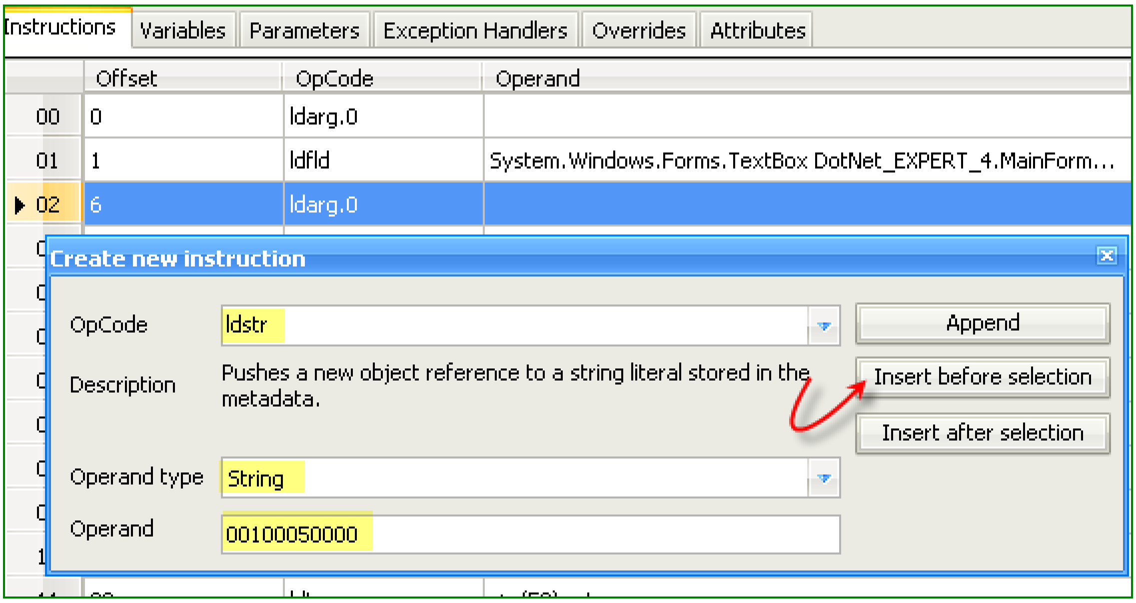


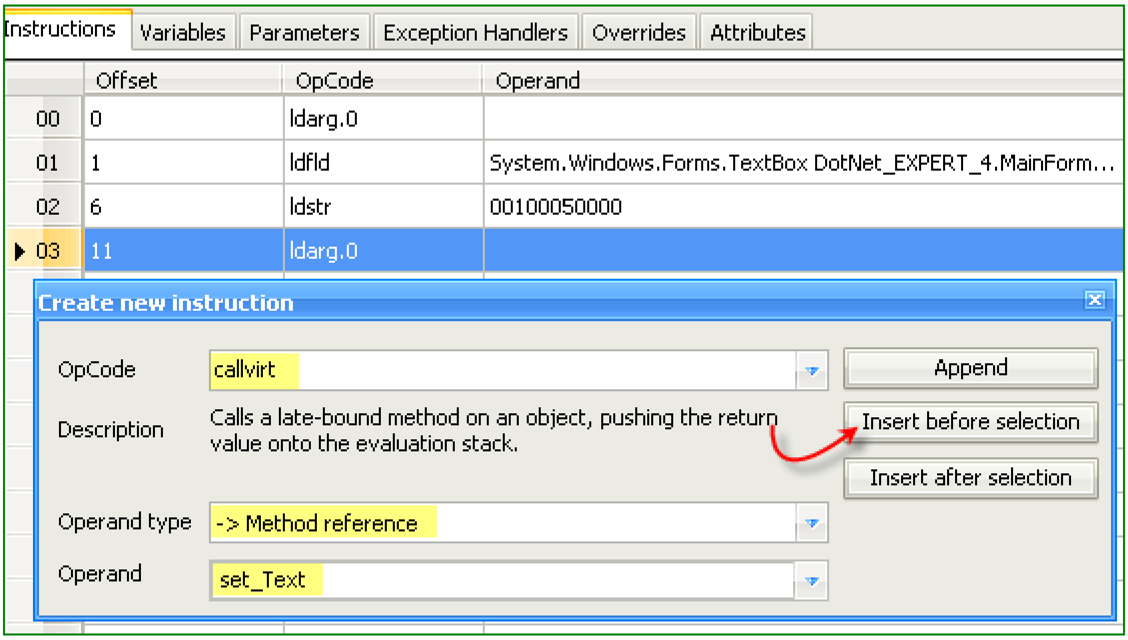


الآن إسناد قيمة للمربع النصي Txt3 (المرجعية (Ldarg.0) + تحميل المربع النصي (Ldfld) + تحميل النص (LdStr) + إسناد النص للمربع النصي (Set_Text) :

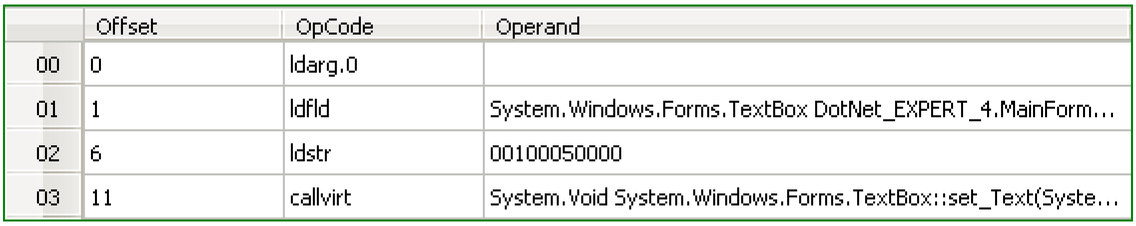








الشكل النهائي للإضافات :



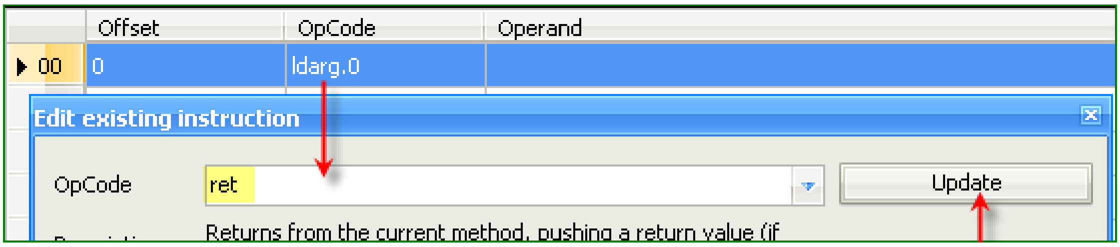


حمل الملف الذي حفظته و شاهد التعديلات :

```
Public Sub Txt3TextChanged(ByVal sender As Object, ByVal e As EventArgs)
  Me.Txt3.Text = "00100050000"
  If ((Strings.Len(Me.Txt3.Text) > 10) AndAlso (Me.MasterID >= 1)) Then
    Dim str As String = (Strings.Mid(Me.Txt3.Text, 3, 1) & Strings.Mid(Me.Txt3.Text, 7, 1))
```

5. خامساً: تجاوز الحدث Timer

أفضل الطرق للتأكد من عمل الإجراءات السابقة بشكل صحيح هي إلغاء دور الحدث Timer من خلال إضافة التعليمة Ret إلى بداية الأجزاء.





6. سادساً: إيجاد السريال للقسم Serial Fishing

إذا فتحنا الحدث Btn3Click سنجد سلسلة طويلة من التعليمات، وهي لا تهمنا، فما يهمنا هي هذه الأسطر:

```
Dim str As String = "DotNetExpert4StepSerialFishingOnly_NoPatching"
```

```
num = 1
Do While (num <= num3)
    num2 = (num2 + Strings.InStr(str, Strings.Mid(Me.txtn.Text, num, 1), CompareMethod.Binary))
    num += 1
Loop
If (num2 = 0) Then
    Interaction.MsgBox("InCorrect Serial...            ", MsgBoxStyle.Exclamation, Nothing)
Else
```

هنا يحاول البرنامج أخذ حروف الاسم, ويجمع قيمة مواقعها في المتحول (Str) إذنً يجب الانتباه إلى أن يكون الاسم محتوياً على حرف واحد من السلسلة (Str) على الأقل؛ لأنه إذ لم يكن أحد حروف الاسم موجوداً في السلسلة Str فعندها سترجع الدالة InStr القيمة 0 وتخرج رسالة الخطأ :

```
Dim num6 As Integer = Strings.Len(Me.txts.Text)
num = 1
Do While (num <= num6)
    str5 = (str5 & Conversion.Hex(Strings.Asc(Strings.Mid(Me.txts.Text, num, 1))))
    num += 1
Loop
If (("41543452452D" & str4) = str5) Then
    Interaction.MsgBox(" Good Work ... !! ", MsgBoxStyle.OkOnly, Nothing)
                                                                        Nothing)
```

قيمة ثابتة | بقية السريال الصحيح بالهيكس | ناتج تحويل السريال المدخل إلى هيكس

كما تلاحظ في الصورة : السريال الصحيح = القيمة الثابتة (D41543452452) + قيمة المتحول Str4

لنحلل القيمة الثابتة :

Hex	41	54	34	52	45	D2
Ascii	A	T	4	R	E	–

إذنً : الكلمة الثابتة = (**AT4RE-**)

والآن لنحاول إيجاد السريال للاسم (**Roben**) باستخدام المنقح Deblector ولإيجاد قيمة المتحول Str4 :

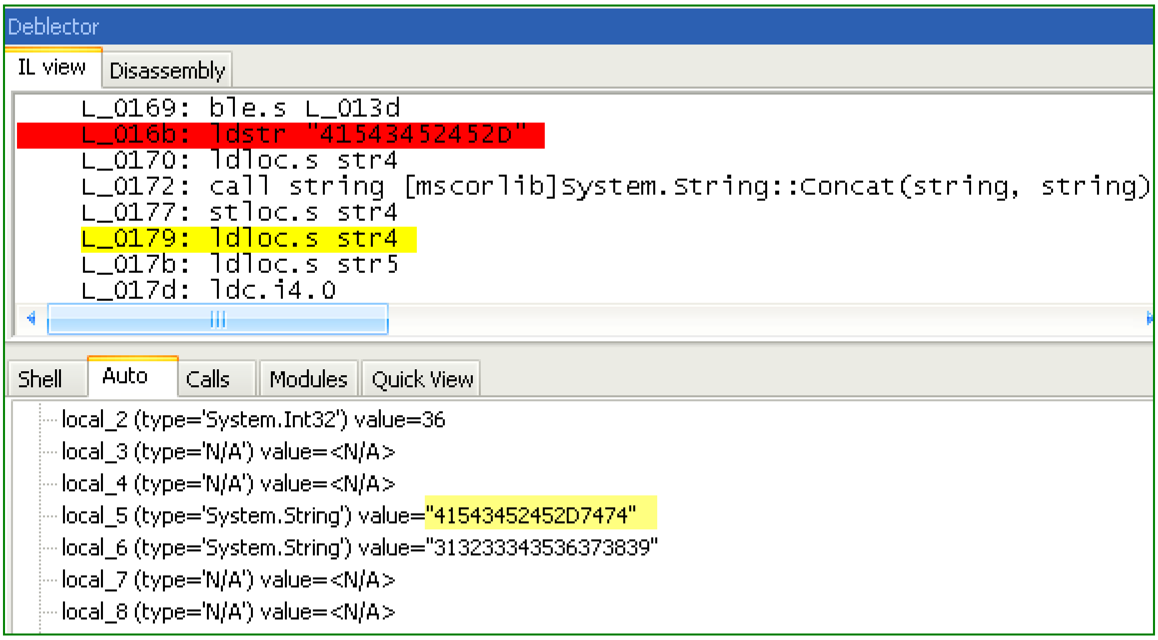
نحمل البرنامج بالمنقح كما تعلمنا، ونضع نقطة توقف عند السطر L_016b , ثم ندخل البيانات في البرنامج؛ مثلاً : الاسم : Roben ، السريال : 123456789 نضغط بعدها Check Serial يتوقف المنقح عند نقطة التوقف السابقة.

نضغط F10 عدة مرات ونلاحظ في اللوحة Auto رقمنا المدخل بالهكس :

313233343536373839 = 123456789

مع السريال الصحيح + القيمة الثابتة (بالهكس) :

41543452452D7474



tt = 7474

إذن السريال الصحيح للاسم Roben هو : AT4RE-tt .

7. سابعاً: عمل كيجن للقسم Advanced وشرح الـ Brute Force Attack

7.1 دراسة الكود

لنتصفح الكود الموافق للضغط على الزر Bt4Click (OK) :

كما تلاحظ في الصورة التالية : يجب أن يكون طول الاسم : 12 ، وطول السريال : 15

```
Public Sub Bt4Click(ByVal sender As Object, ByVal e As EventArgs)
  If ((Strings.Len(Me.TN.Text) <> 12) AndAlso (Strings.Len(Me.TS.Text) < 15)) Then
    Interaction.MsgBox("InCorrect Input...            ", MsgBoxStyle.Exclamation, Nothing)
```

بعدها يتم أخذ أول 7 حروف من السريال ويتم تحويلها إلى أحرف كبيرة :

```
ElseIf Not GenR.MX(GenR.GX(Strings.Mid(Me.TS.Text, 1, 7).ToUpper)) Then
  Interaction.MsgBox("InCorrect Serial...            ", MsgBoxStyle.Exclamation, Nothing)
```

ويتم تمرير الأحرف السبعة السابقة إلى الدالة GX ضمن الوحدة GenR :

```
ElseIf Not GenR.MX(GenR.GX(Strings.Mid(Me.TS.Text, 1, 7).ToUpper)) Then
   Interaction.MsgBox("InCorrect Serial...            ", MsgBoxStyle.Exclamation, Nothing)
```

سندخل الآن إلى الدالة GX ونحلل ما يحدث :

نلاحظ تمرير الأحرف السبعة الأولى إلى هاشينغ SHA1 :

```
Public Shared Function GX(ByVal StrIn As String) As String
   Dim left As String = ""
   Try
      Dim algorithm As HashAlgorithm = New SHA1CryptoServiceProvider
      Dim bytes As Byte() = Encoding.UTF8.GetBytes(StrIn)
      Dim buffer As Byte() = algorithm.ComputeHash(bytes)
```

يجب الانتباه هنا إلى أن Buffer هو مصفوفة من البايتات تتراوح قيمة كل حد بين (000 – 255) أي كل حد مؤلف من 3 أرقام، و لكن في الرياضيات : الصفر على اليسار مهمل (مثلاً : 78 = 078 , 4 = 004).

السطر التالي يقوم بجمع حدود المصفوفة Buffer ويفصل بينها بواسطة الشرطة (– (ويعيد ناتج العملية على شكل نص :

```
Dim i As Integer = 0
Do While (i <= num2)
   left = Conversions.ToString(Operators.ConcatenateObject(Operators.ConcatenateObject(left, Interaction.IIf((i = 0), "", "-")), buffer(i).ToString))
   i += 1
Loop
```

ماذا سيحدث لناتج تمرير الحروف السبعة الأولى من السريال إلى الدالة GX ؟ سيتم تمريرها للدالة MX :

```
ElseIf Not GenR.MX(GenR.GX(Strings.Mid(Me.TS.Text, 1, 7).ToUpper)) Then
   Interaction.MsgBox("InCorrect Serial...            ", MsgBoxStyle.Exclamation, Nothing)
```

ماذا تحوي هذه الدالة؟ نذهب لها ونلاحظ أنها دالة مقارنة مع القيمة StrM الموجودة في cctor :

```
Public Shared Function MX(ByVal StrIn As String) As Boolean
   Return (StrIn = GenR.StrM)
End Function
```

الخلاصة لما سبق :

دفع الحروف السبعة الأولى من السريال لهاشينغ SHA1 ينتج سلسلة نصية يفصل بين حدودها الشرطة (–)

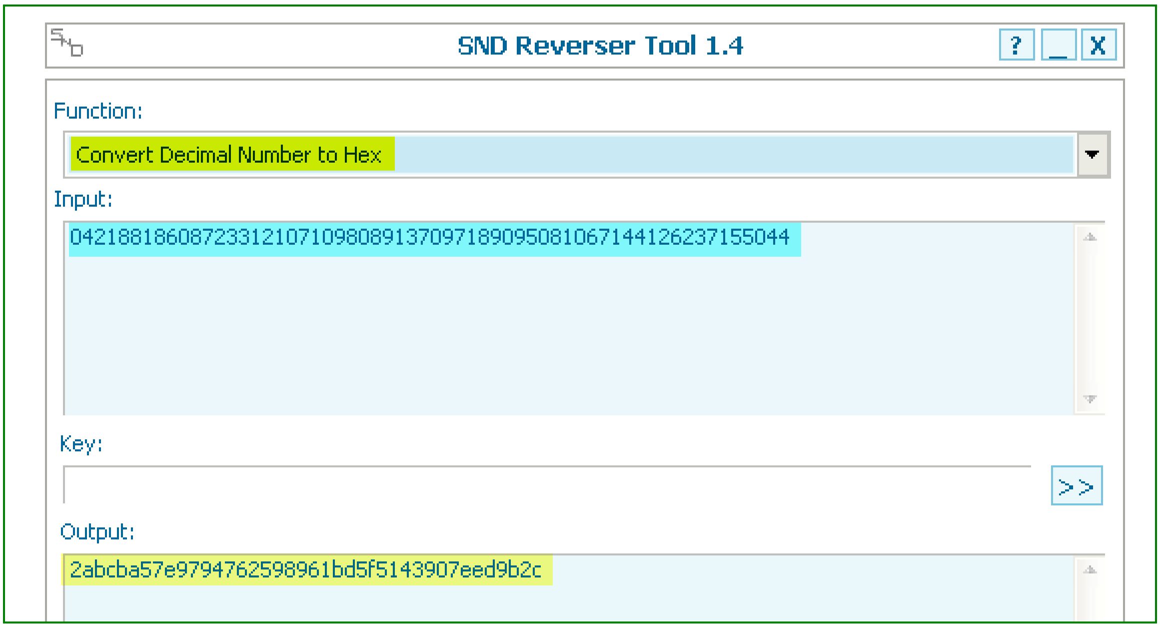
(كل حد مؤلف من 3 أرقام والأصفار في بداية كل حد تحذف)

تقارن السلسلة الناتجة مع السلسلة النصية (StrM)

– لنعيد صياغة StrM بحذف الشرطات وإضافة الأصفار للحدود الناقصة :

StrM	42	188	186	87	233	121	71	98	89	137	97	189	95	81	67	144	126	237	155	44
Repair	042	188	186	087	233	121	071	098	089	137	097	189	095	081	067	144	126	237	155	044
RESULT	042188186087233121071098089137097189095081067144126237155044																			

– ولكن كما نعلم فإن الـ SHA1 تنتج قيمة من النوع Hex إذاً لنحول Result إلى Hex من خلال SnD RT :



إذاً : المعلومات التي بين يدينا الآن :

الحروف السبعة الأولى من السريال تحول إلى أحرف كبيرة.

يعمل لها هاشينغ SHA1 .

ناتج الهاشينغ للحروف السبعة الأولى هو: abcba57e9794762598961bd5f5143907eed9b2c2

السؤال المطروح هو : ما هي الحروف السبعة الكبيرة التي هاشها بـ SHA1 هو : abcba2 ..... ؟

حقيقة للإجابة عن هذا السؤال يلزمنا استخدام تقنيات الـــ Brute Force Attack (BFA) أو ما يعرف باسم الهجوم العشوائي .

و يعني هنا : تجريب جميع الكلمات المؤلفة من 7 حروف وعمل هاشينغ SHA1 لها، ومقارنة الناتج مع الهاش الذي لدينا.

7.2 SHA بروت فورسينغ

سنستخدم لعمل BFA طريقتين :

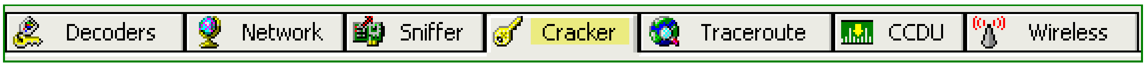
الطريقة الأولى : استخدام برنامج **Cain** (تجده في هذا الرابط ).

الطريقة الثانية : استخدام برنامج **Shabr** (مرفق مع الدرس).

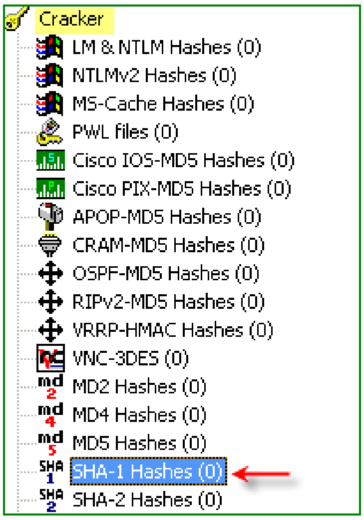
7.2.1 الطريقة الأولى : استخدام برنامج Cain

الطريقة العادية

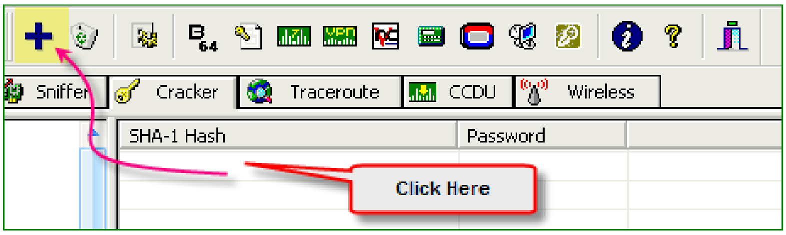
– نفتح برنامج Cain ونتوجه إلى الصفحة (Cracker) :



– نختار من القائمة اليسرى الهاش SHA1 :

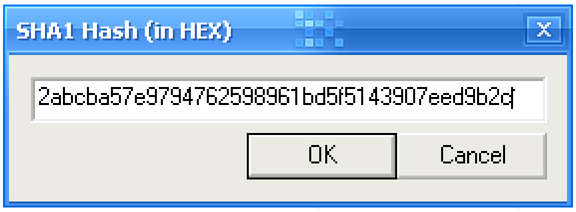


– ثم نضغط في أحد الحقول الفارغة فتفعل الزر (+) :

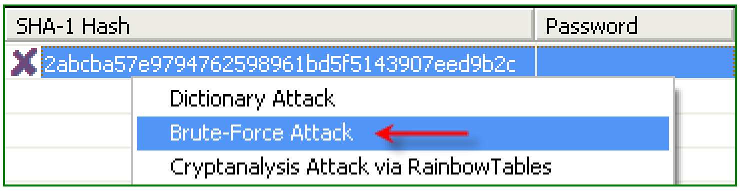


– يظهر لنا مربع حواري ندخل ضمنه الهاش الذي لدينا :

abcba57e9794762598961bd5f5143907eed9b2c2



– فيدرج الهاش الذي لدينا ضمن القائمة، الآن نضغط بزر الفأرة الأيمن ونختار Brute Force Attack :



– يخرج لنا نموذجاً جديداً، نعين القيم في النموذج كالتالي :

Min : يمثل أصغر طول للكلمة المراد البحث عنها (كون الكلمة لدينا مؤلفة من 7 حروف إذن : سنجعلها 7) .

Max : يمثل أكبر طول للكلمة المراد البحث عنها (كون الكلمة لدينا مؤلفة من 7 حروف إذن : سنجعلها أيضاً 7) .

Charset : يمثل سلسلة المحارف التي تحتويها كلمتنا :

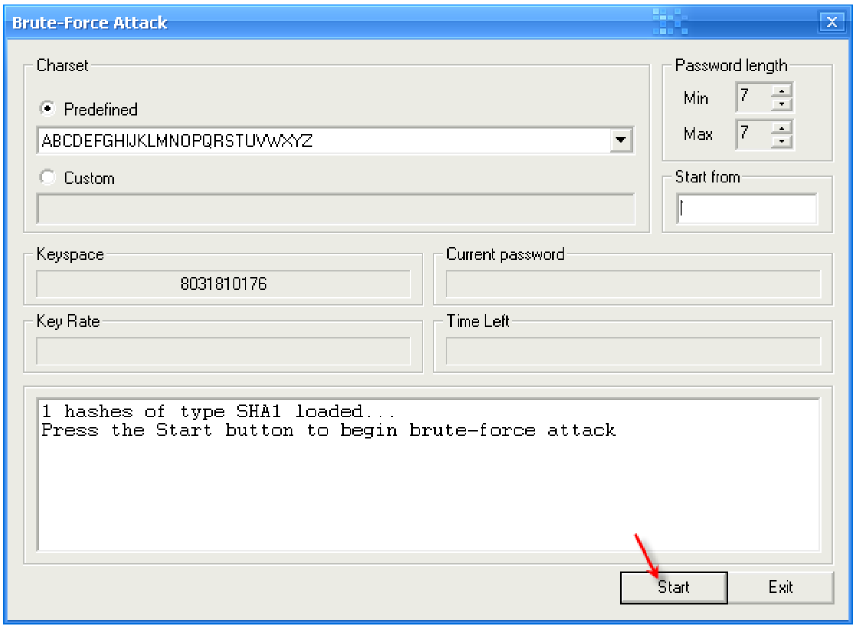
– كون كلمتنا تُحول إلى أحرف كبيرة : فهي إما أحرف كبيرة فقط أو أحرف كبيرة وأرقام أو أحرف كبيرة وأرقام ورموز.

– وسنبدأ الهجوم بالاحتمال الأقل (أحرف كبيرة فقط) وإذا لم يفلح ننتقل إلى الاحتمال الثاني فالثالث .

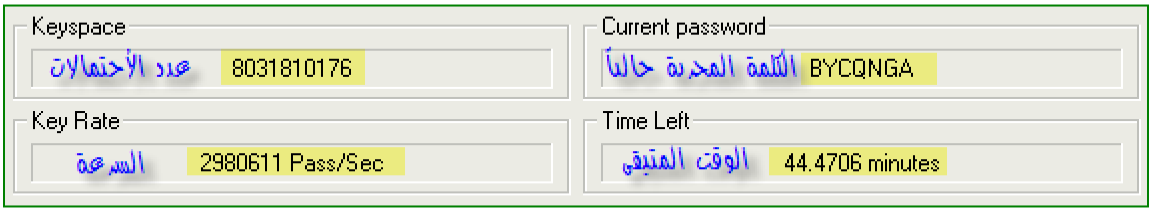
Custom : يستخدم إذا كنت على علم بما قد تتكون منه الكلمة، أو إذا أردت تخمين بعض الكلمات؛ مثلاً:

(**AT4REROBENDOTNETEXPERT4**) وإذا نجح تخمينك فستصل إلى الكلمة الصحيحة بسرعة .

– ولكننا سنعتمد أولاً المبدأ التقليدي لأنه الأشمل، قم بملء الحقول كما في الصورة :

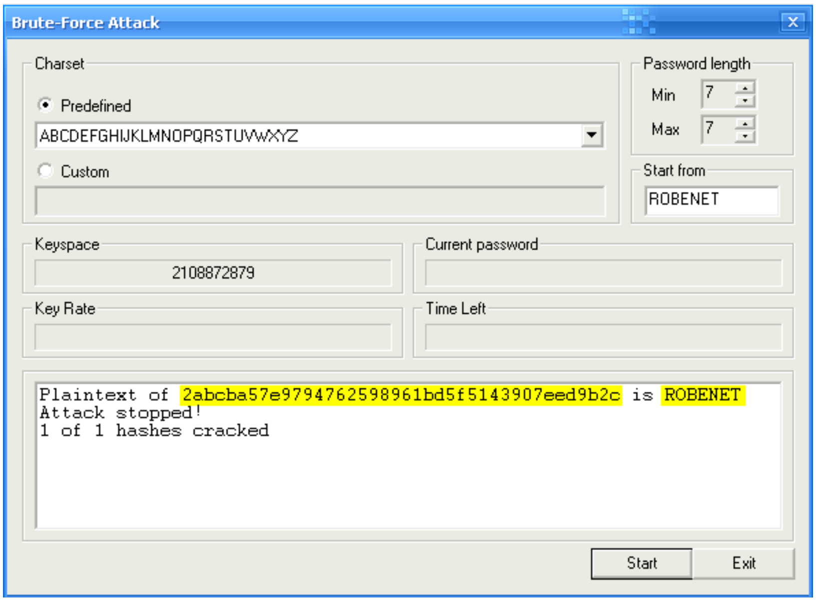


– يبدأ العمل :



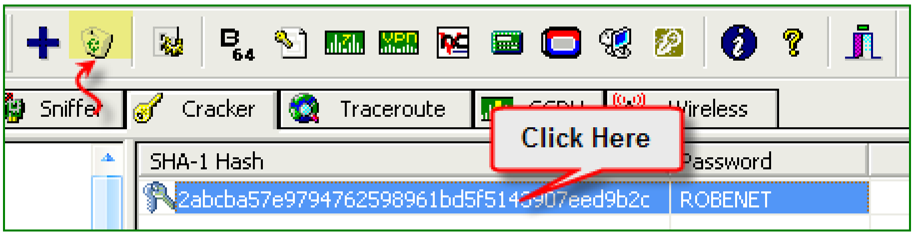
– يجب الانتباه إلى أن معدل السرعة والوقت المتبقي يختلف من جهاز إلى آخر حسب مواصفات كل جهاز.

– بعد مضي حوالي (30 دقيقة) تم اكتشاف الكلمة المقابلة للهاش المدخل، وهي (ROBENET) :



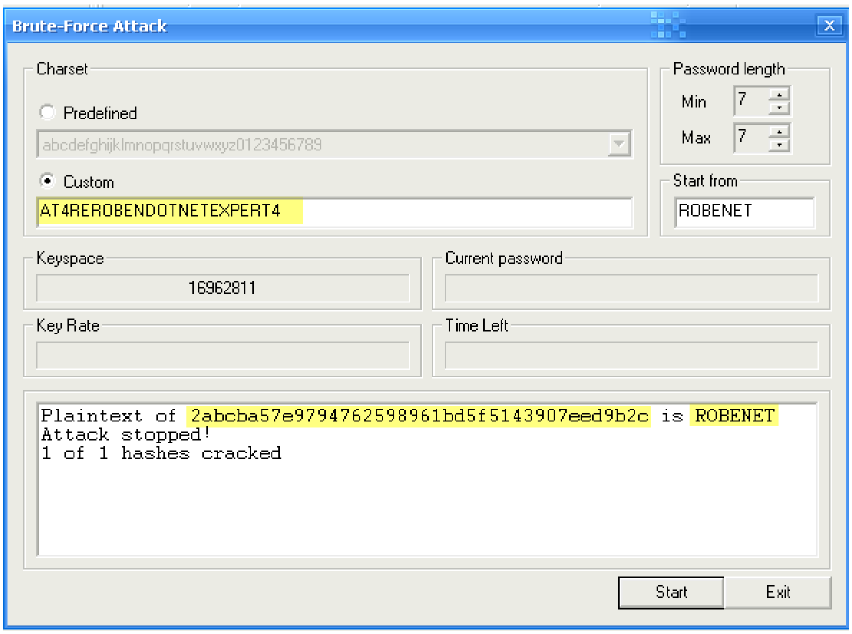
طريقة التخمين :

– أغلق النموذج السابق واضغط على الهاش المدخل واحذفه بالضغط على زر السلة :



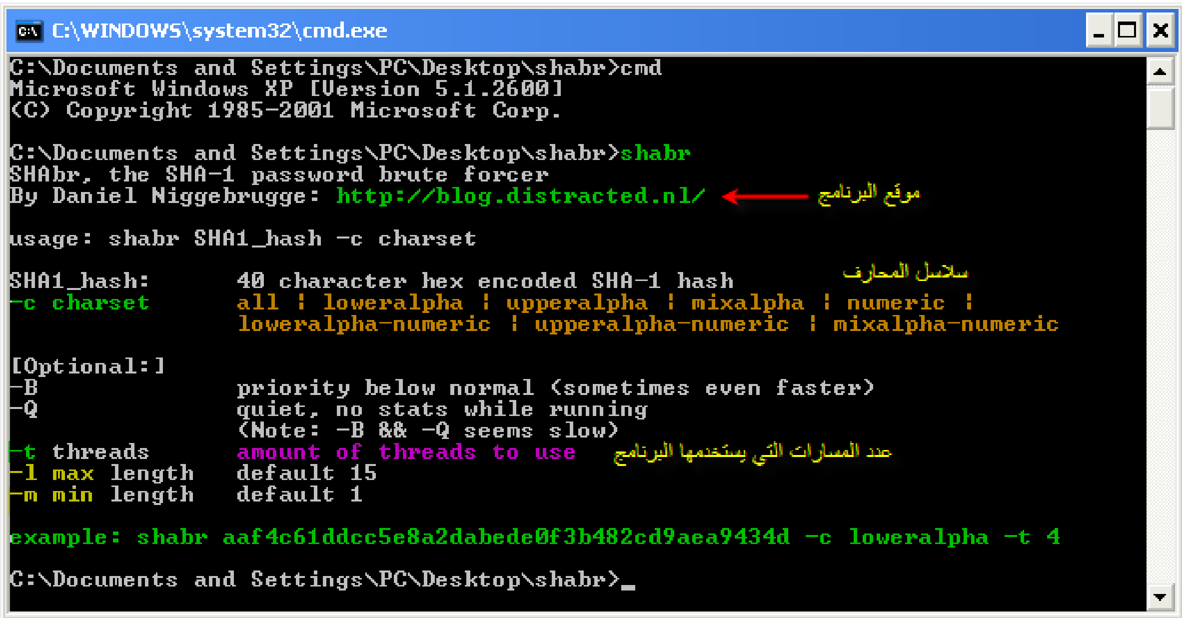
– أعد إدخال الهاش كما فعلنا مسبقاً ثم زر اليمين واختر Brute Force Attack والفرق الوحيد هنا أننا سنكتب السلسلة (AT4REROBENDOTNETEXPERT4) في الحقل Custom كما في الصورة ثم Start :

– نلاحظ اكتشاف الكلمة بسرعة وأن المدة قد تقلصت إلى (3 ثوانٍ) :



7.2.2 الطريقة الثانية : استخدام برنامج Shabr

نشغل الملف الدفعي RunMe.Bat ونكتب الأمر التالي (Shabr) ثم Enter للتعرف على باراميترات البرنامج وطريقة استخدامه :



اكتب سطر الأوامر التالي ثم اضغط Enter (إذا كان جهازك متواضعاً قلل الــ t مثلاً : إلى 2 أو 4) :

Shabr 2abcba57e9794762598961bd5f5143907eed9b2c –c UpperAlpha –t 8 –m 7

```
C:\Documents and Settings\PC\Desktop\shabr>cmd
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\PC\Desktop\shabr>Shabr 2abcba57e9794762598
07eed9b2c -c upperalpha -t 8 -m 7
```

نلاحظ بدء العمل :

```
C:\Documents and Settings\PC\Desktop\shabr>cmd
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\PC\Desktop\shabr>Shabr 2abcba57
07eed9b2c -c upperalpha -t 8 -m 7
SHA-1 hash:      2abcba57e9794762598961bd5f5143907eed9b2c
Threads:         8
Lower priority:  no
Quiet:           no
Length:          7-15
Charset:         upperalpha
ABCDEFGHIJKLMNOPQRSTUVWXYZ

Length 7 - 4% in 47.14 s (7.50 Mhashes/s)
```

بعد مضي حوالي (600 ثانية   حوالي 10 دقائق) وجد البرنامج الكلمة الهدف وهي ROBENET :

```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\PC\Desktop\shabr>cmd
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\PC\Desktop\shabr>Shabr 2abcba57e97947
07eed9b2c -c upperalpha -t 8 -m 7
SHA-1 hash:      2abcba57e9794762598961bd5f5143907eed9b2c
Threads:         8
Lower priority:  no
Quiet:           no
Length:          7-15
Charset:         upperalpha
ABCDEFGHIJKLMNOPQRSTUVWXYZ

Length 7 - 61% in 655.02 s (7.49 Mhashes/s)

>>> Found: ROBENET <<<

Length 7 - 61% in 655.72 s (7.49 Mhashes/s)
C:\Documents and Settings\PC\Desktop\shabr>
```

تعليق:

- استغرق Shabr ثلث المدة التي استغرقها Cain (ولكن استخدامه لذاكرة الجهاز أكبر).
- استغرقت طريقة التخمين 3 ثوانٍ فقط, ولكن هذا الأمر لا ينجح دائماً.

إذنً الحروف السبعة الأولى من السريال هي (**ROBENET**).

نتنقل الآن إلى السطر التالي من الحدث Bt4Click:

نلاحظ تمرير الاسم المدخل إلى الدالة G3X :

```
ElseIf Not GenR.G3X(Me.TN.Text) Then
   Interaction.MsgBox("InCorrect Serial...          ", MsgBoxStyle.Exclamation, Nothing)
```

لندخل إلى الدالة ولنرى ماذا يفعل بالاسم؟ (لا حظ الشرح على الصورة) :

```
Public Shared Function G3X(ByVal StrIn As String) As Boolean
   Dim flag As Boolean
   Dim str3 As String = "596573476F4D616E"
   Dim str4 As String = "50-69-54-69-54-53-55-52-52-53-55-56-55-48-51-52-50-68-53-57-54-53-55-51-52-5
   Dim expression As String = ""
   Dim str2 As String = ""
   Dim str5 As String = ""
   expression = Strings.Mid(StrIn, 1, 9)
   str2 = Strings.Mid(StrIn, 10, 3)
   Dim num3 As Integer = Strings.Len(expression)
   Dim i As Integer = 1
   Do While (i <= num3)
      str5 = (str5 & Conversion.Hex(Strings.Asc(Strings.Mid(expression, i, 1))))
      i += 1
   Loop
   str5 = GenR.G2X((str5 & str3))
   Try
      If (str5 = str4) Then
         flag = True
      Else
         Return False
      End If
   Catch exception1 As
      ProjectData.SetProje
      Dim exception As E
      ProjectData.ClearProjectError
   End Try
   If Not Versioned.IsNumeric(str2) Then
      Return False
   End If
   Return flag
End Function
```

الحروف التسعة الأولى من الأسم تحول إلى هيكس

يضاف لهكس الأسم قيمة الثابت
Str3 = 596573476F4D616E
و يمرر الناتج للدالة:
G2X

G2X يقارن ناتج الدالة مع الثابت :
Str4 = 50-69-54 ......

الحروف الثلاث الأخيرة من الأسم يجب أن تكون أرقام

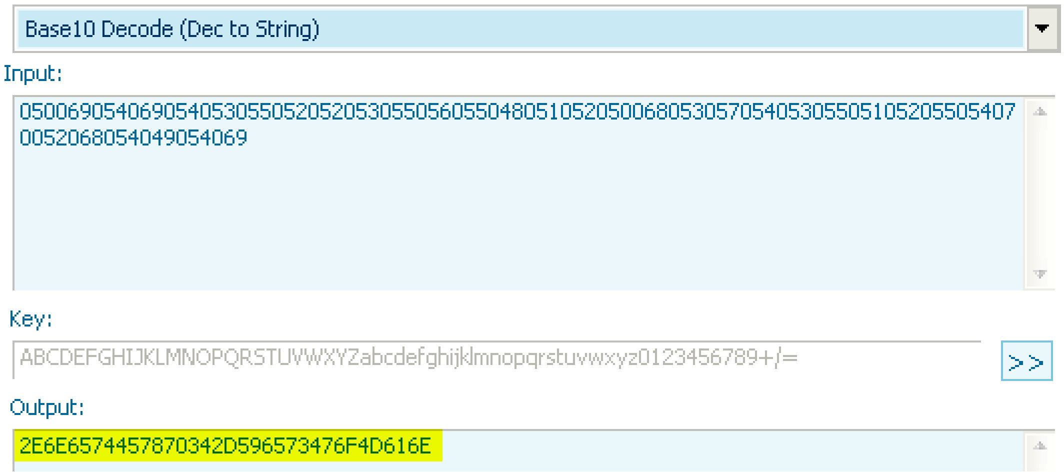
إذا دخلنا إلى الدالة G2X سنجدها نفس تقنية التحويل التي تكلمنا عليها سابقاً (مصفوفة يفصل بينها شرطات) ولكن بدون هاشينغ :

```
Public Shared Function G2X(ByVal StrIn As String) As String
    Dim left As String = ""
    Try
        Dim bytes As Byte() = Encoding.UTF8.GetBytes(StrIn)
        Dim num2 As Integer = (bytes.Length - 1)
        Dim i As Integer = 0
        Do While (i <= num2)
            left = Conversions.ToString(Operators.ConcatenateObject(Operators.ConcatenateObject(left, Interaction.IIf((i = 
            i += 1
        Loop
```

لنحول Str4 كما تعلمنا (حذف الشرطات وإضافة الأصفار) ينتج لنا :

05006905406905405305505205205305505605504805105205006805305705405305505105205505407005206805404905406905

نحول القيمة السابقة إلى String باستخدام SnD RT :



القيمة الناتجة هي : **E6E6574457870342D596573476F4D616E2**

لا حظ أن القسم المعلم بالأحمر هو : Str3= **596573476F4D616E**

إذن القسم المعلم بالأزرق هو الحروف التسعة الأولى من الاسم المعرف : Str5 = **2E6E6574457870342D**

وهو يقابل بالـــــ ASCII :

(Str5(Hex	E2	E6	65	74	45	78	70	34	D2
ASCII	.	n	e	t	E	x	p	4	-
Result	.netExp4-								

الخلاصة :

الاسم المعرف مكون من : .netExp4– مضاف إليها أي 3 أرقام (مثال : .netExp-478. .... netExp4- 001).

الحروف السبعة الأولى من السريال هي : ROBENET .

لنتابع في السطر التالي للإجراء Bt4Click :

يتم أخذ الحروف الثمانية الأخيرة من السريال لتمرر للدالة G4X :

```
ElseIf Not GenR.G4X(Strings.Mid(Me.TS.Text, 8, 8)) Then
    Interaction.MsgBox("InCorrect Serial...              ", MsgBoxStyle.Exclamation, Nothing)
```

لندخل إلى الدالة G4X ولنشاهد ما تفعله بالحروف الثمانية الأخيرة من السريال (لاحظ الشرح على الصورة التالية) :

```
Public Shared Function G4X(ByVal StrIn As String) As Boolean
    If Not Versioned.IsNumeric(StrIn) Then
        Return False
    End If
    Dim num As Integer = 0
    Dim num2 As Integer = 0
    Dim str As String = "0891011150711110770 97110"
    Dim num3 As Integer = (Strings.Len(StrIn) - 1)
    num = 0
    Do While (num <= num3)
        num2 = (num2 + Strings.InStr(str, Conversions.ToString(StrIn.Chars(num)), CompareMethod.Binary))
        num += 1
    Loop
    If (num2 < 24) Then
        Return False
    End If
    Return True
End Function
```

يجب أن تكون الحروف لثمانية الأخيرة (أرقام)

يجمع مواقع الأرقام الثمانية في السلسلة Str

يجب أن يكون المجموع ٢٤ فما فوق

المجموع يجب أن يكون 24 فما فوق.

يجب الانتباه هنا إلى أن الدالة StrIn تعيد موقع أول عنصر مطابق (في حال كان هناك أكثر من عنصر مطابق مثل : 0,1,9,7).

نلاحظ العناصر المكونة لــ Str :

- 0 : موقعه = 1
- 1 : موقعه = 4
- 7 : موقعه = 11
- 8 : موقعه = 2
- 9 : موقعه = 3

إذن سنستبعد الصفر لصغر قيمته، وسنشكل الحروف الخمسة الأولى من (1789).

- إذا أردنا توليد الرقم بشكل عشوائي يجب أن نضع باعتبارنا تكرار أصغر قيمة (مثلاً : القيمة : 8).
- في الحروف الثلاثة الباقية سنحتاج أرقاماً ذات موقع كبير؛ مثل (7 ,1 , 9)، ومثل: **88899179** .

7.3 كتابة الكيجن

سنحاول كتابة الكيجن معتمدين على ما يلي :

الاسم مؤلف من (.netExp4-) + (أي ثلاثة أرقام).

السريال مؤلف من (ROBENET) + (خمس أرقام مؤلفة من الأرقام 1789) + (ثلاثة أرقام مؤلفة من الأرقام 179).

سنستخدم دلفي لكتابة الكيجن :

– قم بإنشاء مربعي نص باسم (TXTN – TXTS)، وزراً باسم (BtnGen)، واكتب الكود التالي :

```
Function RandomX( IntIn: Integer ): String;
Const
StrArry1 : Array [0..3]Of String = ('1','7','8','9');
StrArry2 : Array [0..2]Of String = ('1','7','9');
StrArry3 : Array [0..9]Of String =('0','1','2','3','4','5','6','7','8','9');
Var Str: String; I : Integer;
begin
Str:= '';
Randomize;
if IntIn = 5 Then
    begin For I:= 1 To 5 Do
          Begin
          Str := Str + StrArry1[Random(4)];
          End;
          Result := Str;
    End
Else  If IntIn = 31 Then
     begin For I:= 1 To 3 Do
          Begin
          Str := Str + StrArry3[Random(9)];
          End;
          Result := Str;
     End
Else  If IntIn = 32 Then
     begin For I:= 1 To 3 Do
          Begin
          Str := Str + StrArry2[Random(2)];
          End;
          Result := Str;
     End;
End;

procedure TForm1.BtnGenClick(Sender: TObject);
begin
```

```
TXTN.text := '.netExp4-' + RandomX(31);
TXTS.text := 'ROBENET' + RandomX(5) + RandomX(32);
end;
```

- في الحقيقة حاولنا اختصار الكود أكثر (بشرط توليد قيمة عشوائية)، ولكننا لم نتوصل إلى طريقة أقصر من ذلك.
- من لديه طريقة أقصر من ذلك فليشاركنا بها لنتناقش حولها .



كلمة ختامية :

– نتمنى أن تكونوا قد استمتعتم واستفدتم من قراءة هذا الدرس، ونرجو من الله تعالى أن يجعله خالصاً لوجهه الكريم .

– إذا كان لديكم أي استفسارات أو أسئلة نرجو طرحها بحرية، وسنسعد بالإجابة عنها.

أمثلة تطبيقية لكسر برامج دوت نت

.Net RCE Examples

سيكون هذا الدرس بمثابة تطبيق عملي لما تعلمته سابقاً عن تحليل برامج الدوت نت، ولنبدأ على بركة الله :

I. إيجاد السريال لتطبيقات الدوت نت : Serial-fishing in DotNet Targets

سنقدم هنا عرضاً سريعاً لتنقيح برمجيات الدوت نت بهدف إيجاد السريال باستخدام الإضافة المميزة لبرنامج Reflector والتي تدعى Deblector .

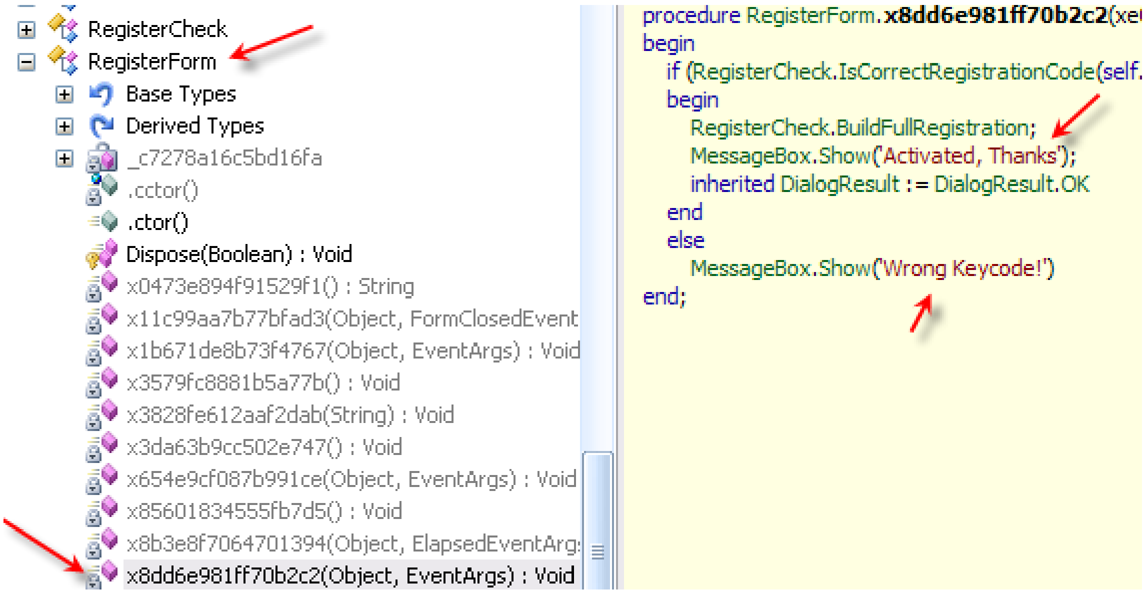


- Reflector
- Deblector

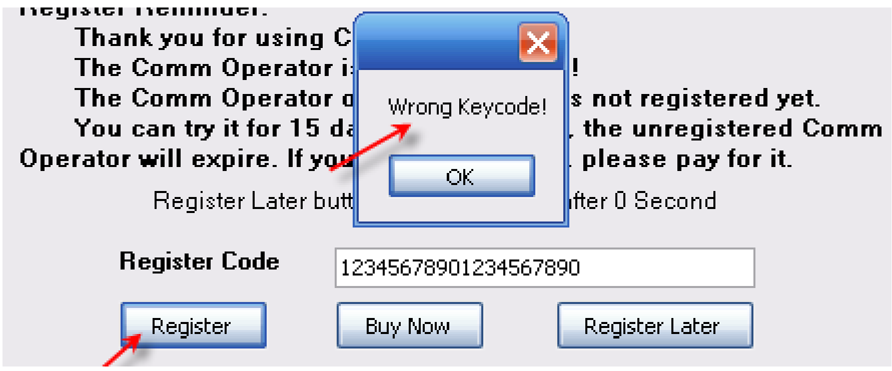


Comm Operator

نحمل البرنامج داخل برنامج Reflector ثم ننتقل إلى العقدة CommOp نتصفح العناوين, سيجذب نظرنا مباشرة العنوان : RegisterForm ندخل إليه ونتصفح الإجراءات بداخله، نجد أسماء الإجراءات مشوشة, لا بأس نتصفحها بالترتيب, ويتوقفنا هذا الإجراء :



وهو يمثل الإجراء المستخدم عند الضغط على الزر Register ونرى رسالة التسجيل الصحيح ورسالة الخطأ :

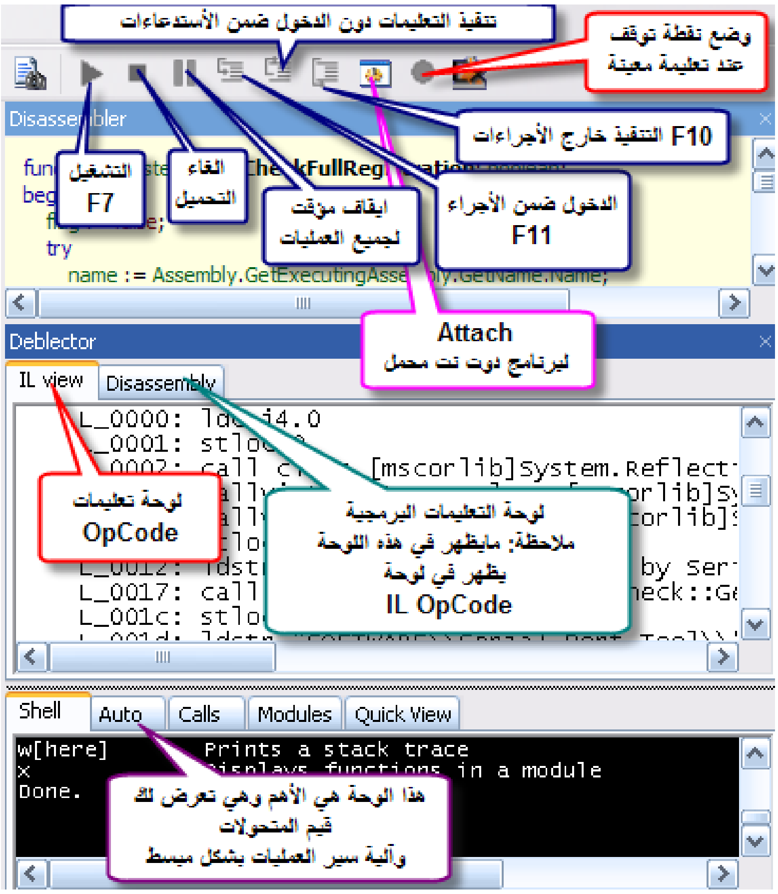


يمكن إيجاد السريال لهذا البرنامج بتتبع الإجراءات المستخدمة، ولكن هذا ليس هدف الدرس, على العموم سنقدم عرضاً سريعاً لهذه الطريقة في نهاية الشرح.

نحمل الإضافة Deblector من القائمة Tools والتي ستقوم بدور المنقح في برمجيات الدوت نت :



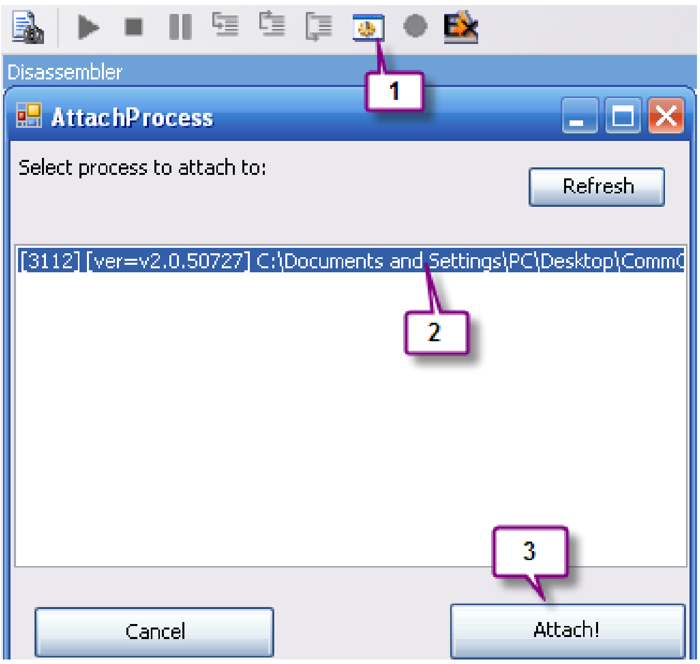
ظهر لنا شريط الأدوات الخاص بالمنقح في الأعلى, ولوحات البرنامج في الجزء السفلي :



خطوات عمل المنقح :

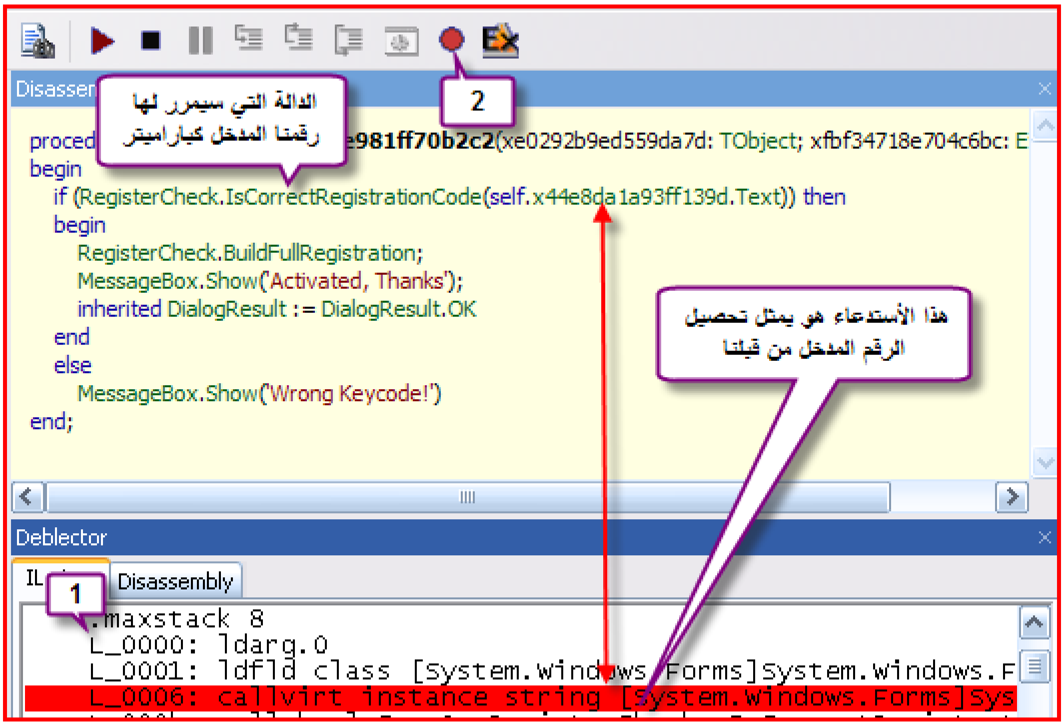
1) تشغيل البرنامج المراد تنقيحه.

2) ضغط على زر Attach واختيار البرنامج.



3) وضع نقطة توقف عند تعليمة معينة لتتبع سير العملية (يجب الانتباه إلى أن المنقح يجب أن يكون في حالة توقف مؤقت لتتمكن من وضع نقطة توقف، ويمكن الضغط على زر التوقف المؤقت Pause لوضع نقطة توقف أثناء تشغيل البرنامج).

في مثالنا سنضع نقط توقف عند بداية إجراء الضغط على الزر Register عند السطر L0000 بوضع المؤشر في اللوحة IL View ثم الضغط على زر نقطة التوقف أو الضغط على المفتاح F9 :



– الآن نطلق المنقح بالضغط على F7 (Start) وندخل رقم تسجيل معين، وليكن **1234567890** ثم نضغط على زر التسجيل Register.

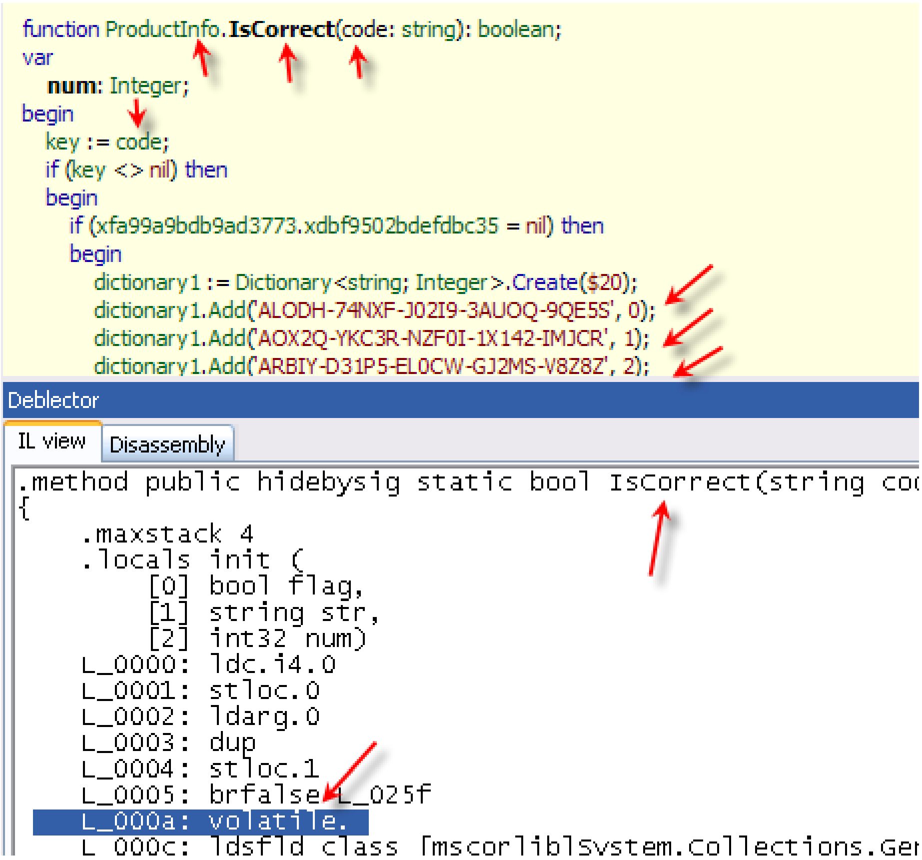
– يتوقف المنقح عند السطر L0000 للأسف Deblector منقح تفتقر إلى المرونة، وهنالك جوانب نقص كثيرة فيه، وهذا أمر طبيعي فهو ما يزال في طور التطوير.

– اضغط على F11 أكثر من مرة للدخول ضمن الاستدعاء الذي تريده، كما يمكنك استخدام الأمر Step Out للخروج السريع من الإجراء الحالي.

– يعتبر إضافة نقط توقف مدروسة عاملاً مهماً في تنقيح جيد ودقيق للبرنامج، ويبعدك عن الدخول في عشرات الأسطر من الكود.

– بعد توقف المنقح عند السطر L0000 نضغط F11 عدد من المرات، ندخل بعدها إلى إجراء فرعي (غير هام بالنسبة لنا), استخدم الآن الزر Step Out للخروج من الإجراء الحالي، ستدخل في إجراء جديد أيضاً غير مهم هنا، اضغط Step Out مرة أخرى, افعل ذلك مرة أخرى، وعندها سنعود إلى إجراءنا الرئيسي.

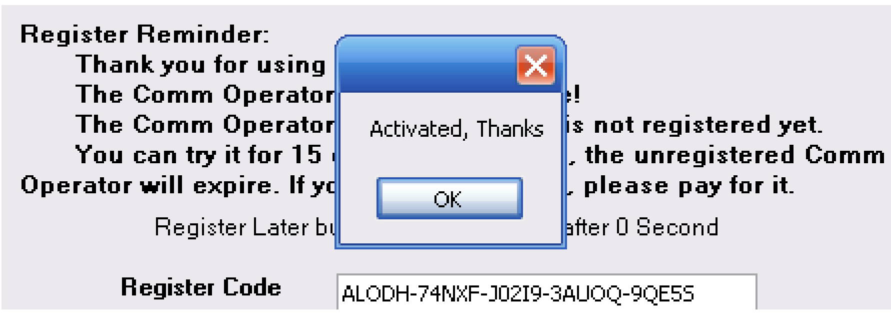
– الآن أضغط على F11 وستجد نفسك هنا :



– كما تلاحظ يقوم البرنامج بمقارنة الكود المدخل Code مع مصفوفة من القيم، وفي حال التطابق يعيد القيمة True ، لنجرب إحدى القيم، وليكن :

ALODH-74NXF-J02I9-3AUOQ-9QE5S

نلاحظ أن البرنامج قبِل الرقم وأصبح مسجلاً بالكامل :



– حقيقة ما أعجبنا هو السريال : ILoveSPT يبدو أن مبرمجنا يتمتع بحس الفكاهة :

في الختام :

ذكرنا في المقدمة أن السريال يمكن إيجاده بسهولة باتباع الضغط على الإجراءات والدالة للوصول إلى الصفحة التي وجدناها من خلال المنقح) ولكن كان الهدف التعرف على المنقح وكيفية التعامل معه (.

– اتبع الصور التالية بالنقر حسب السهم :

```
if (RegisterCheck.IsCorrectRegistrationCode(self.x44e8da1a93ff139d.Text)) then
begin
```

```
function RegisterCheck.IsCorrectRegistrationCode(code: string): boolean;
begin
  Result := ProductInfo.IsCorrect(code)
end;
```

```
function ProductInfo.IsCorrect(code: string): boolean;
var
  num: Integer;
begin
  key := code;
  if (key <> nil) then
  begin
    if (xfa99a9bdb9ad3773.xdbf9502bdef[illegible]c35 = nil) then
    begin
      dictionary1 := Dictionary<string; Integer>.Create($20);
      dictionary1.Add('ALODH-74NXF-J02I9-3AUOQ-9QE5S', 0);
      dictionary1.Add('AOX2Q-YKC3R-NZF0I-1X142-IMJCR', 1);
      dictionary1.Add('ARBIY-D31P5-EL0CW-GJ2MS-V8Z8Z', 2);
      dictionary1.Add('ACKTB-HVRDM-2POQN-R07YY-O0SW1', 3);
      dictionary1.Add('A7WNQ-JL5XI-S4T6D-0A5A8-YGOW1', 4);
      dictionary1.Add('AQF5L-K09ZC-P6YV2-6BD6V-RED1G', 5);
      dictionary1.Add('AC4E5-2QHB8-XJZPW-VZJZI-JQGEI', 6);
      dictionary1.Add('AI6LS-OKCND-FR80U-XEJSM-ZNI6O', 7);
      dictionary1.Add('AC3EV-H8SPL-U7M49-4VY8J-DLI8Z', 8);
```

تصل هنا

2. الباتشينغ في تطبيقات الدوت نت : Patching DotNet Targets

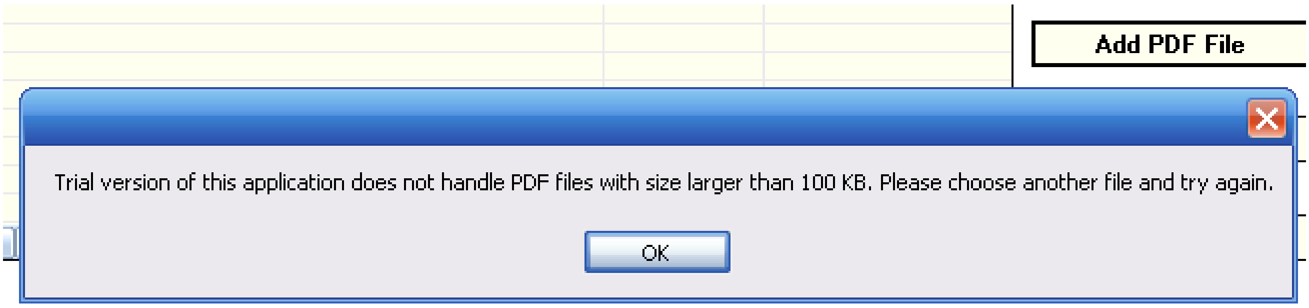
سنقوم هنا بشرح عمل باتشينغ بسيط لأحد تطبيقات الدوت نت من خلال تحليل الكود ببرنامج Reflector

- Reflector
- Reflexil V 0.9 (Reflector

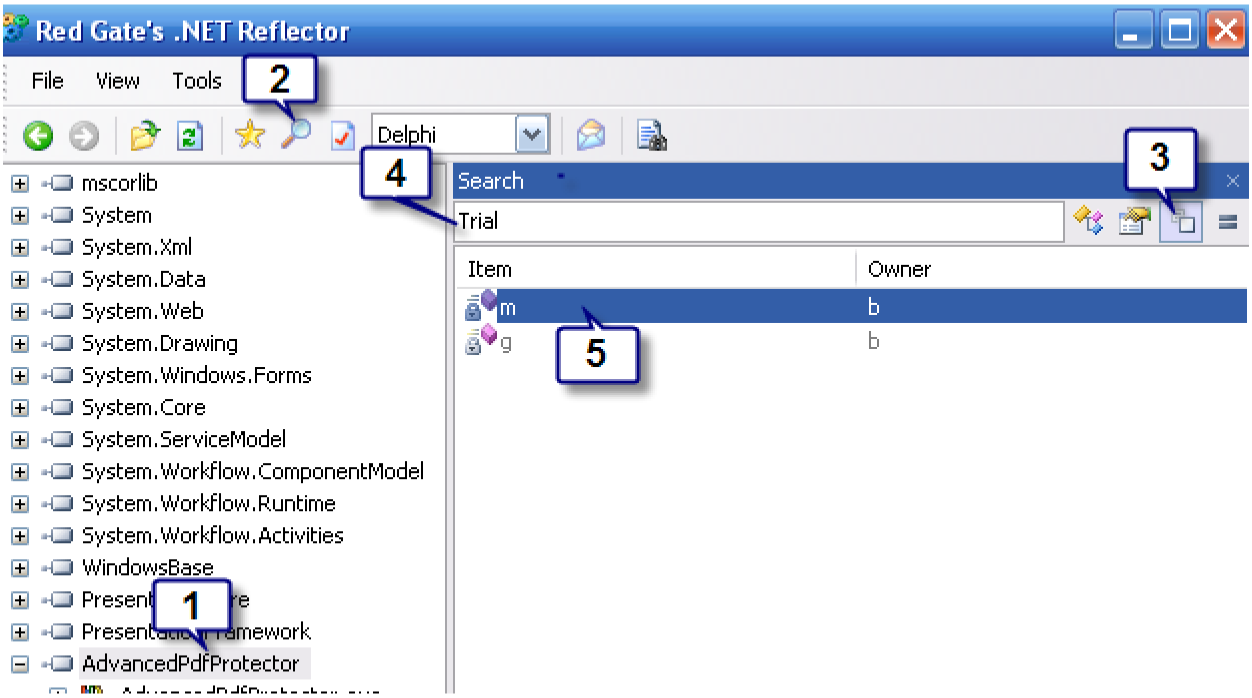
Advanced PDF Protector, Version 1.1

نشغل البرنامج ونتفحصه لمعرفة محدودية البرنامج.

اضغط الزر (Add pdf File) واختر ملفاً بحيث يكون حجمه أكثر من 100 كيلوبايت، عندها سيرفض البرنامج إدراج الملف وسيخبرك بأن النسخة التجريبية محدودة بـــ 100 كيلوبايت فقط :

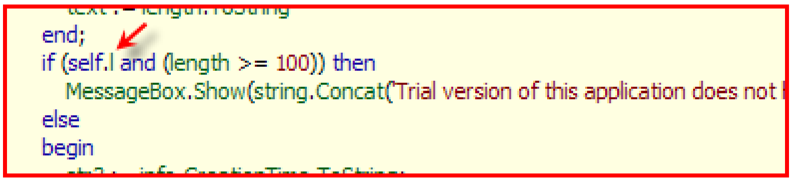


ننطلق الآن إلى برنامج Reflector ونسقط الملف بداخله (البرنامج مرفق)، نختار الملف ثم نضغط على أمر البحث، ونختار الزر Search Strings وندخل الكلمة Trial الموجودة في الرسالة :



وجد لنا البرنامج عنصرين يحتويان على الكلمة Trial التي تفيد بأن النسخة تجريبية، لندخل إلى العنصر m

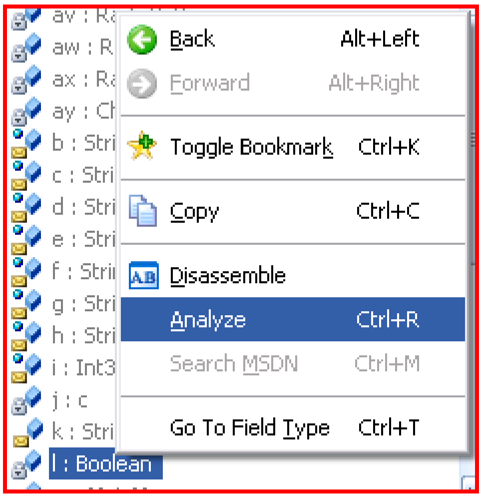
ثم نضغط بشكل مزدوج على m: void سيظهر لنا الكود كالتالي :



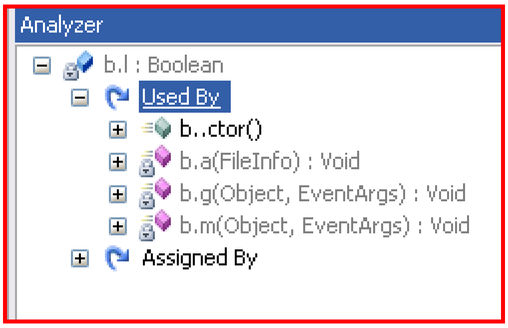
إذا قرأت الكود جيداً ستلاحظ أن قيمة المتغير (L) هي من تحدد أن النسخة تجريبية أم لا؟

لا حظ هنا: إذا كانت قيمة L = True فالبرنامج نسخة تجريبية، أما إذا كانت L= False فالبرنامج نسخة مسجلة.

لنضغط على المتغير L لمعرفة أين تستند له القيمة، بعد الضغط نقر بزر الفأرة الأيمن ونختار الأمر Analyze :

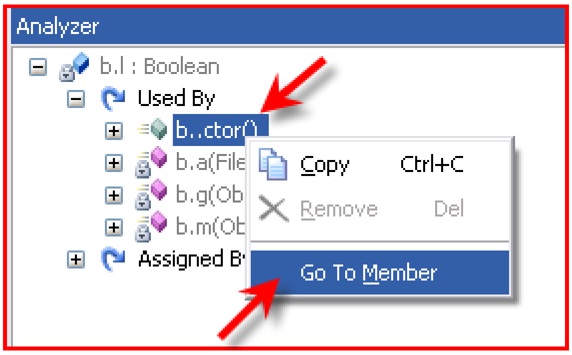


نختار من اللوحة Analyze العقدة Used By لنتتبع ولنعرف أين يستخدم المتحول؟ وأين تستند له القيمة True or False ؟



اختر الأول B.Ctor بالضغط عليه بزر الفأرة الأيمن واختر من القائمة Goto to member .

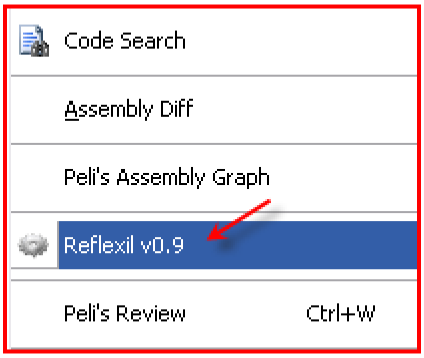
في الحقيقة هذا الإجراء هو مقصودنا، ويمكنك الاطلاع على التحقق لمزيد من التحليل، ولن ندخل إليها حتى لا يطول الشرح.



عند الدخول للإجراء ستلاحظ كيف يتم إسناد قيمة المتحول L بناء على شرط مسبق :

```
constructor b.Create;
begin
   self.a;
   self.k := k.a;
   if (not k.a(self.k, string.Empty)) then
      self.l := true
   else
      self.l := false;
```

يكفينا عكس الجملة الشرطية ليصبح البرنامج مسجلاً بالكامل، وسنستخدم هنا الأداة المساعدة Reflexil وهي إحدى إضافات البرنامج Reflector ، نفتح الإضافة من القائمة Tools :



يعبر عن الشرط السابق باللغة الوسيطة (IL) بالإجراء brtrue.s والذي يعني برمجياً :

If ..... Then

True

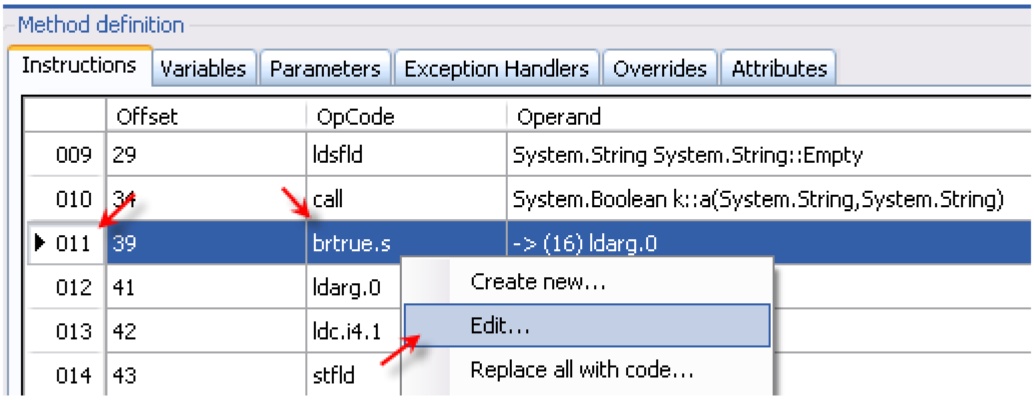
Else

False

ولعكسه يجب استخدام الإجراء brfalse.s ويعني برمجياً :

If … then False else True

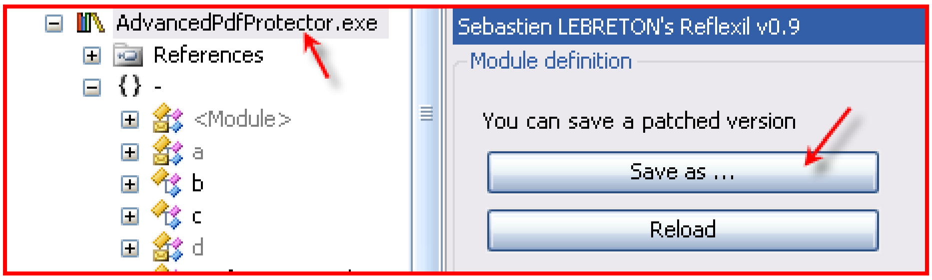
ستجد الإجراء brtrue.s في السطر 011 نضغط عليه بالزر الأيمن ونختار الأمر Edit :



نختار من القائمة المسدلة OpCode ثم نختار الإجراء brfalse.s ثم نضغط Update :



الآن سننتقل إلى حفظ التغيرات من خلال اختيار البرنامج من القائمة اليمنى ثم اختيار الأمر Save as من لوحة الإضافة :



لنجرب الملف المعدل الآن AdvancedPdfProtector.Patched.exe ونستخدم ملف Pdf حجمه أكبر من 100 كيلو (لم تعد تظهر الرسالة) نجرب أيضاً الأمر Protect Pdf File (تم العمل بنجاح) .

كلمة أخيرة :

لاحظنا في البرنامج أن أسماء الطرق والإجراءات والمتغيرات مموهة (a,b , .. L, K) وهذا لأن البرنامج محمي بواسطة برنامج التشويش (dotfuscator) .

3. عمل كيجن لتطبيقات الدوت نت [I] Keygenning DotNet Targets

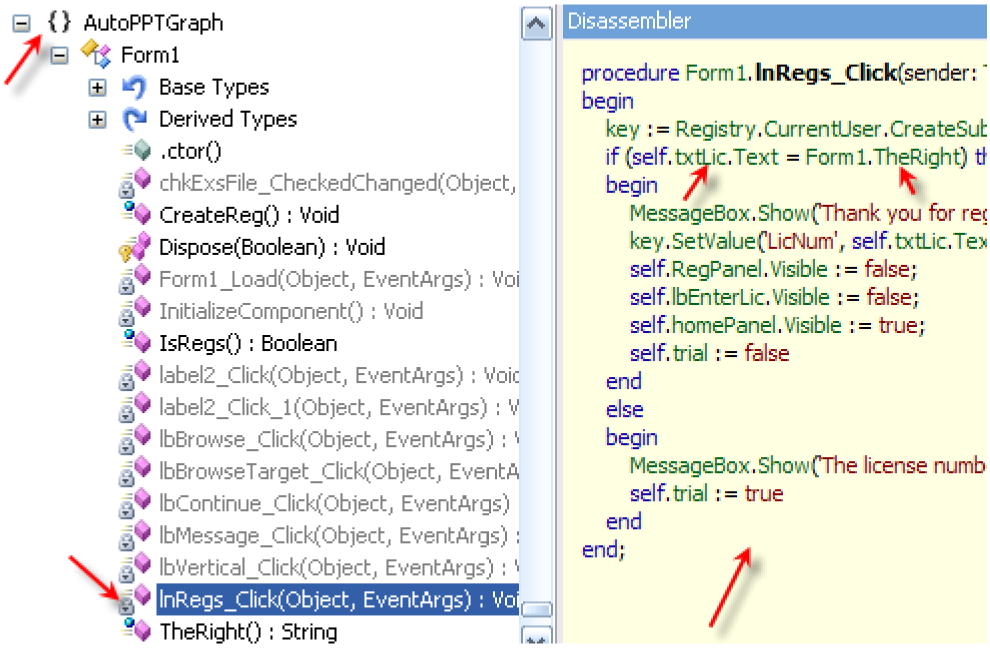
سنقوم في هذا الدرس بعمل كيجن لأحد تطبيقات الدوت نت من خلال قراءة الكود في برنامج Reflector ثم سنقوم بكتابه كيجن للبرنامج باستخدام لغة دلفي.

▪ Reflector ▪ Delphi 7

Autoppt Charts + Keygen src

لنفتح البرنامج باستخدام برنامج Reflector ثم ندخل إلى الملف ونتفحص البرنامج.

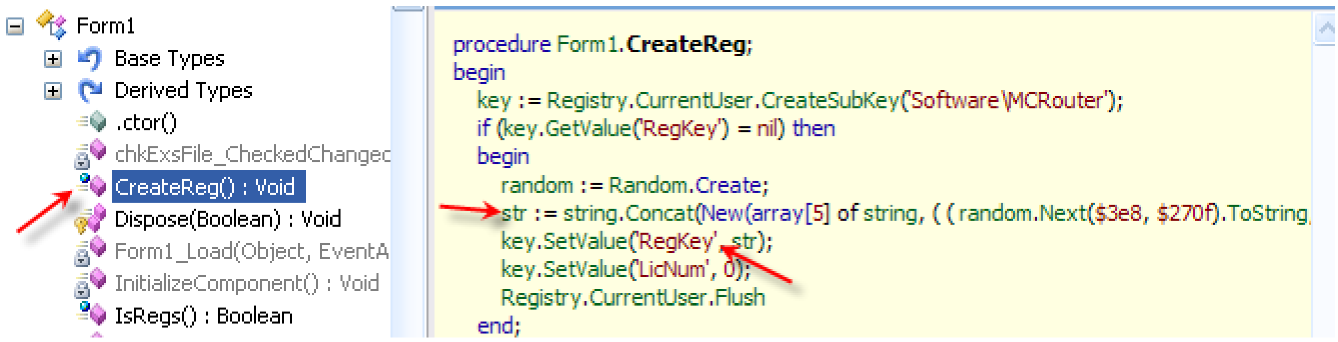
ادخل إلى العقدة AutoPPt Graph وستجد الإجراء **InRegs_Click** اضغط عليه ولنتتبع ما يحدث :



كما تلاحظ هناك مقارنة بين رقمين ونتيجة المقارنة تحدد فيما إذا كان التسجيل صحيحاً أم لا؟

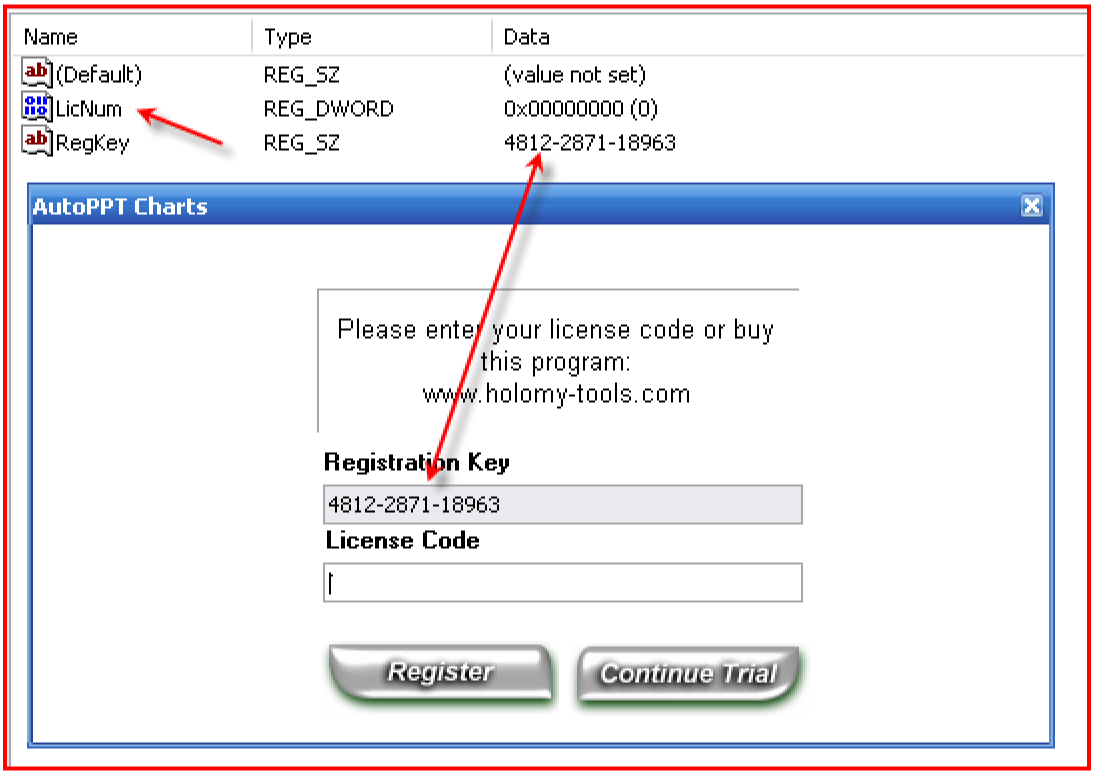
إذن المقارنة تتم بين المتغيرين txtLic و TheRight ولكن ما هما هذان المتحولان؟

لاحظ في اللوحة اليسارية هناك الإجراء CreateReg وهو مثير للاهتمام لندخل ونرى ما يحتويه :



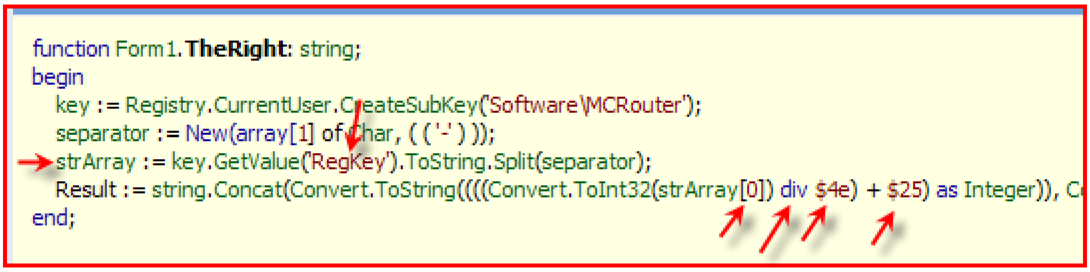
كما تلاحظ يقوم هذا الإجراء بحساب الرقم المعرف للجهاز من خلال الدالة Random ويشكل مفتاحاً معرفاً على الشكل : XXXX-XXXX-XXXX (لاحظ السطر str)

بعد ذلك يقوم بتخزين هذه القيمة في الريجستري في المفتاح الفرعي RegKey :



إذن ما هي قيمة المتغير TheRight ؟

لنضغط على المتغير TheRight من الإجراء **InRegs_Click** ونراقب كيف يتم حساب هذا المتغير :



لندقق معاً في سلسلة حساب السريال المقابل من الرقم المعرف :

1) يؤخذ الرقم المعرف **RegKey** وتزال منه الشرطة (–) ويحول إلى مصفوفة من ثلاثة حدود في حالتنا:

4812-2871-18963 فالحد الأول (4812) الحد الثاني (2871) الحد الثالث (18963)

2) تجرى العملية الحسابية التالية على كل حد ثم ترتب بجانب بعضها :

[Part1 Div (4e (Hex) = 78 (Decimal)] + 25(Hex) = 37(Decimal)

لنطبق على الحدود الثلاثة مع الانتباه إلى أخذ الرقم الصحيح فقط (لاحظ as integer) :

- Part1 : (4812 / 78) + 37 = 61 + 37 = **98**
- Part2 : (2871/78) + 37 = **73**
- Part3 : (18963 / 78) + 37 = **280**
- Serial = Part 1 & Part 2 & Part3 = **9873280**

إذن TheRight يمثل الرقم الصحيح، وهو يقارن مع رقمنا المدخل.

يمكن كتابة الخوارزمية السابقة بدلفي بالشكل التالي :

```
Function GetSerial ( ID : String ) : String;
Var Part1,temp, Part2 ,Part3 ,Serial :String;  I,K :Integer;
Begin
part1:= Copy(ID,1,pos('-',ID)-1);
Temp:=Copy(ID,pos('-',ID)+1, Length(Id));
Part2:= Copy( Temp, 1,pos('-',Temp)-1);
Part3:= Copy( Temp, pos('-',Temp)+1, length ( Temp ));

//-----------------------------------------------------

Part1:= IntToStr(( StrToInt(Part1) Div 78 ) + 37 );
Part2:= IntToStr(( StrToInt(Part2) Div 78 ) + 37 );
Part3:= IntToStr(( StrToInt(Part3) Div 78 ) + 37 );

//----------------------------------------------------

Serial := Part1 + Part2 + Part3 ;
Result:= Serial;
end;
```

9. عمل كيجن لتطبيقات الدوت نت من خلال نسخ الكود وإصلاحه [2] Keygenning DotNet Targets

يوفر لنا برنامج Reflector ميزة مهمة من خلال تحويل الكود من OpCode إلى اللغة البرمجية التي نختارها.

سنحاول خلال هذا الدرس شرح طريقة كتابة كيجن لأحد تطبيقات الدوت نت من خلال نسخ الكود ولصقه ضمن بيئة البرمجة الموافقة للكود المنسوخ .

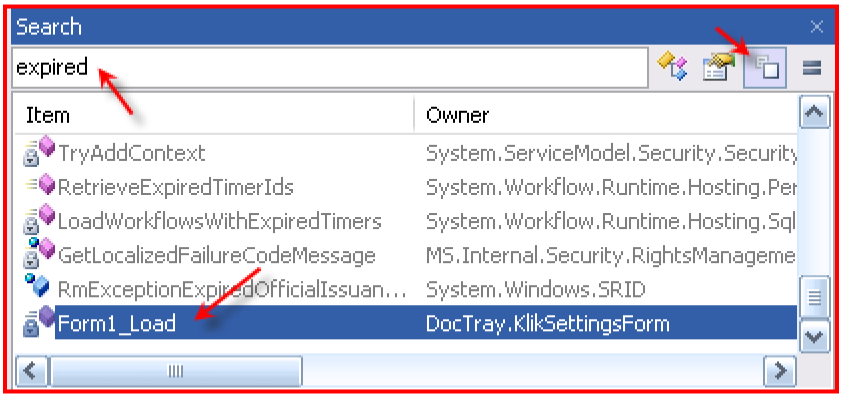
▪ Reflector ▪ Reflexil ▪ SharpDevelop

Doc Tray

– عند تشغيل البرنامج تظهر لنا رسالة تفيد بأن صلاحية النسخة قد انتهت، ويبدو أن المبرمج قرر أن تعمل النسخة حتى تاريخ معين :

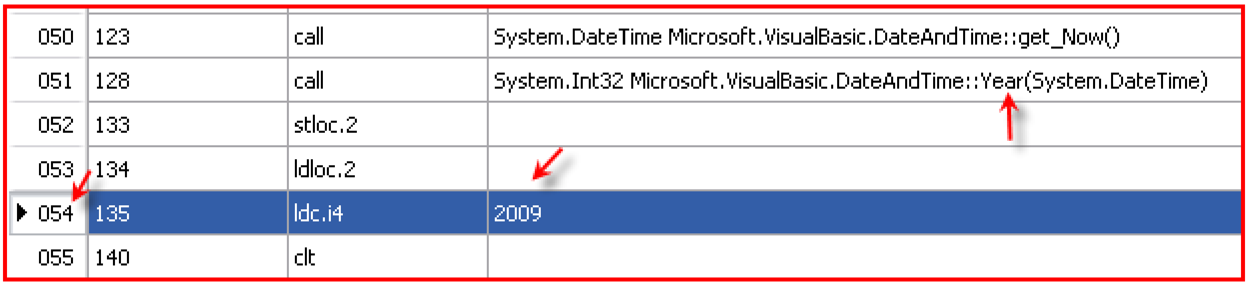


– لنبدأ بتحليل الكود من خلال برنامج Reflector , سندخل في مربع البحث الكلمة expired لنرى أين تم استخدامها؟ (لا تنس اختيار الزر Strings Search) :



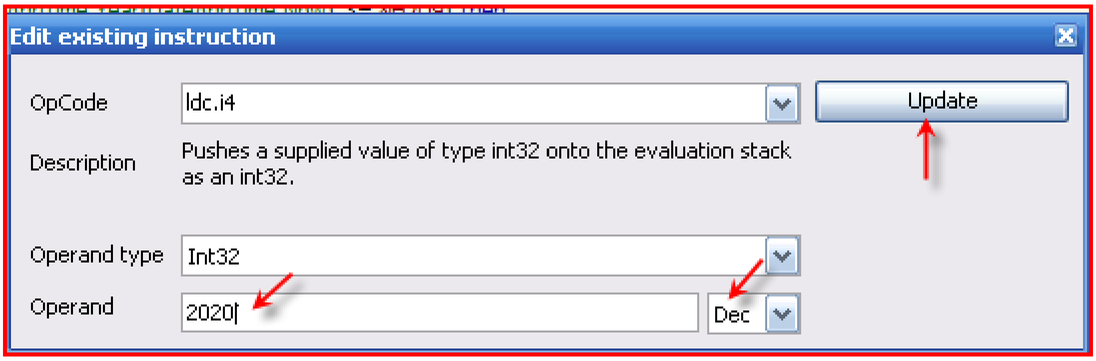
– ندخل إلى الإجراء Form1_Load وسنلاحظ أن المبرمج حدد عمل النسخة قبل عام 2009 =(hex) 79D سنقوم باستخدام الإضافة Reflexil لتغيير العام إلى 2020 مثلاً.

– نفتح الإضافة وننظر في Operand ونجد القيمة 2009 عند السطر 054 :

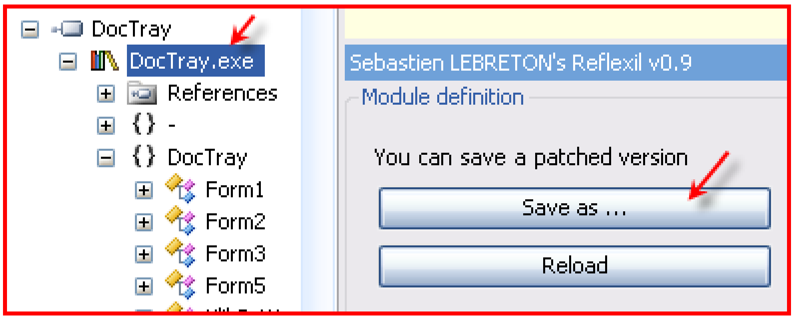


050	123	call	System.DateTime Microsoft.VisualBasic.DateAndTime::get_Now()
051	128	call	System.Int32 Microsoft.VisualBasic.DateAndTime::Year(System.DateTime)
052	133	stloc.2	
053	134	ldloc.2	
054	135	ldc.i4	2009
055	140	clt	

– نضغط على القيمة باليمين ونختار الأمر Edit ثم نحرر القيمة إلى 2020 مثلاً ونضغط Update :



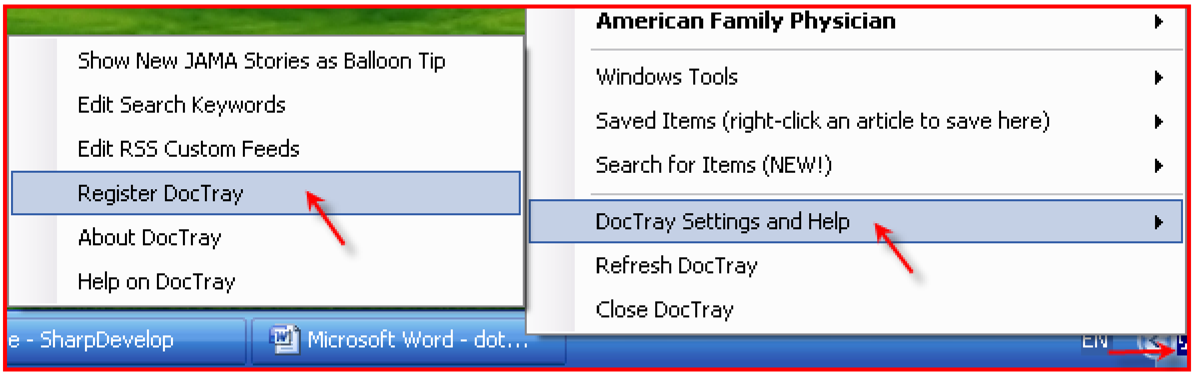
– نحفظ الملف بالضغط على اسم الملف بالأعلى ثم Save as :



– سيحفظ بالاسم : DocTray.Patched.exe في نفس مسار البرنامج.

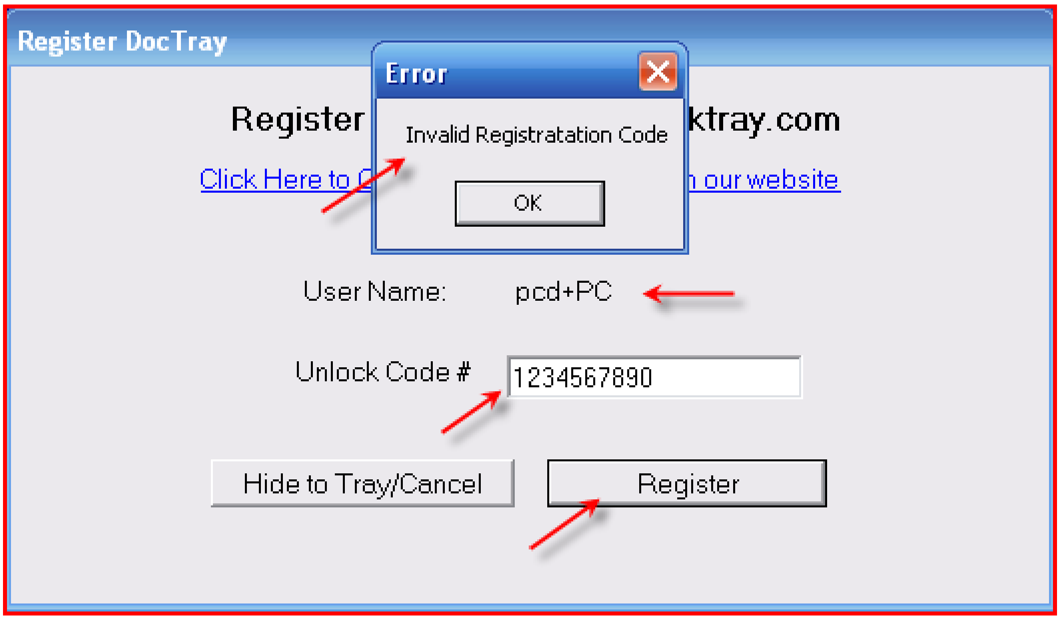
– نغلق الملف القديم , باختياره ثم الزر الأيمن ثم Close , ونحمل الملف المعدل : DocTray.Patched.exe .

– نشغل الملف المعدل، ونلاحظ أن الرسالة اختفت وأن البرنامج وضع بجوار الساعة، نأتي الآن لعملية تسجيل البرنامج، نختار من القائمة الأمر Register Doc Tray :

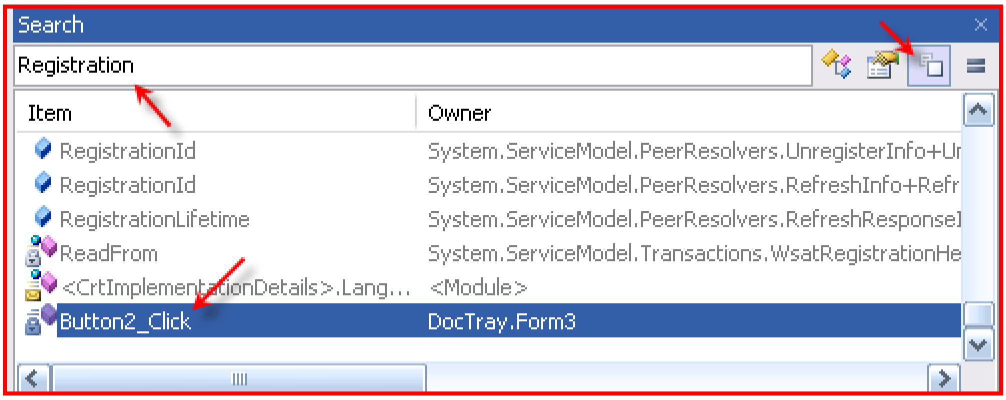


– نلاحظ أن البرنامج يختار رقماً معرفًاللجهاز، إذن فالسريال لا ينفع هنا ونحن بحاجة لكيجن.

– نُدخل أي رقم تسجيل وليكن 1234567890 في الحقل Unlock Code ثم نضغط Register وعندها ستظهر الرسالة :
Invalid Registration Code



– نحاول البحث عن الكلمة Registration وندخل إلى الإجراء Butto2_Click :

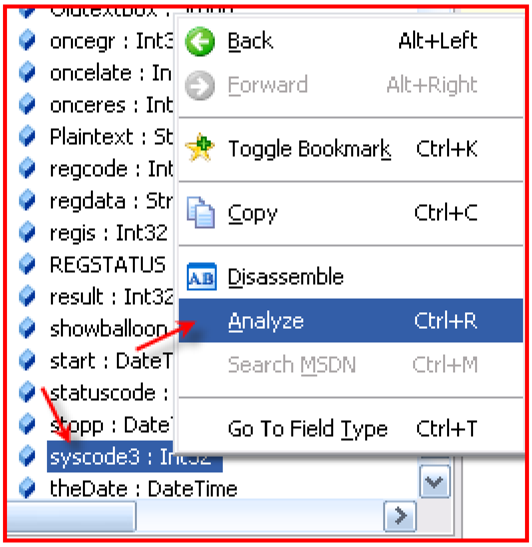


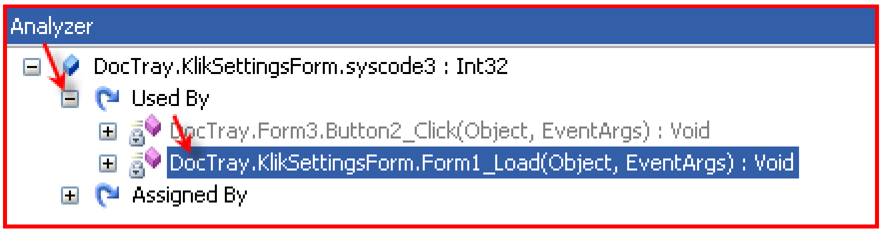
– نلاحظ وجود مقارنة بين رقمنا المدخل والرقم syscode3 :

```
ElseIf (Me.UNLOCK.Text = Conversions.ToString(MyProject.Forms.KlikSettingsForm.syscode3)) Then
    MyProject.Forms.KlikSettingsForm.statuscode = 1
    MyProject.Forms.KlikSettingsForm.regis = 1
```

– نضغط على syscode3 لنبحث عن مرجعيته ومكان استخدامه (يأخذ الأمر بعض الوقت، كن صبوراً!!)

– بعد ظهوره نختر Analyze ثم في اللوحة Analyze نختر العقدة Used By :





– الحدث الأول لا يهمنا (لأننا كنا بداخله قبل قليل)، الحدث الثاني هو المهم Form1_Load لندخل إليه بالضغط بالزر الأيمن ثم على Goto To Member ونتصفح الكود (ملاحظة : نوع لغة البرمجة Visual Basic).

– نلاحظ وجود الكود المسؤول عن توليد الرقم syscode3 هنا :

```
Dim num2 As Integer
Me.regdata = Environment.UserName
If (Strings.Len(Me.regdata) <= 3) Then
    Me.regdata = Dns.GetHostName
End If
If (Strings.Len(Me.regdata) <= 3) Then
    Me.regdata = (Me.regdata & "+" & Environment.UserName)
End If
If (Strings.Len(Me.regdata) <= 3) Then
    Me.regdata = (Me.regdata & "+" & Me.regdata)
End If
If (Me.syscode3 = 0) Then
    num2 = Strings.Len(Me.regdata)
    Me.syscode3 = 0
    Dim num5 As Integer = num2
    Dim i As Integer = 1
    Do While (i <= num5)
        Dim num3 As Integer = Strings.Asc(Strings.Mid(Me.regdata, i, 1))
        Me.syscode3 = (Me.syscode3 + ((&H750 * i) * num3))
        i += 1
    Loop
End If
```

– سنحلل الكود قليلاً ثم سننتقل إلى تحويله لكيجن :

```
Dim num2 As Integer
//تحصيل اسم المستخدم للجهاز
Me.regdata = Environment.UserName
//هل اسم المستخدم مؤلف من 3 حروف أو اقل؟
If (Strings.Len(Me.regdata) <= 3) Then
//إذن حصل اسم الكمبيوتر
Me.regdata = Dns.GetHostName
End If
//هل مازالت القيمة أقل من 4 حروف؟
If (Strings.Len(Me.regdata) <= 3) Then
//إذن أضف اسم المستخدم للقيمة لسابقة
Me.regdata = (Me.regdata & "+" & Environment.UserName)
End If
//هل -رغم ذلك- مازالت القيمة أقل من 4 حروف؟
If (Strings.Len(Me.regdata) <= 3) Then
//إذن كرر القيمة مرتين
Me.regdata = (Me.regdata & "+" & Me.regdata)
End If
If (Me.syscode3 = 0) Then
//إسناد طول القيمة السابقة للمتحول
num2 = Strings.Len(Me.regdata)
Me.syscode3 = 0
Dim num5 As Integer = num2
Dim i As Integer = 1
//الرقم هنا يمثل طول القيمة
Do While (i <= num5)
//أخذ ما يقابل كل حرف بالعشري
Dim num3 As Integer = Strings.Asc(Strings.Mid(Me.regdata,i,1))
//ضرب العداد بـ 1872 ثم ضرب الناتج بـ العشري السابق وجمع قيم الحلقة مع بعضها
Me.syscode3 = (Me.syscode3 + ((&H750 * i) * num3))
i += 1
Loop
End If
```

– لنرى ما يجب علينا تغيره في الكود (لغة البرمجة Visual Basic .net – سنستخدم البرنامج SharpDevelop لكتابة الكيجن) :

1) حذف (.Me) من كل المتغيرات.

2) التصريح عن المتحول regdata كمتحول نصي (String).

3) التصريح عن المتحول syscode3 كمتحول رقمي (Integer).

4) تغير Dns.GetHostName إلى (Environment.MachineName).ToLower.

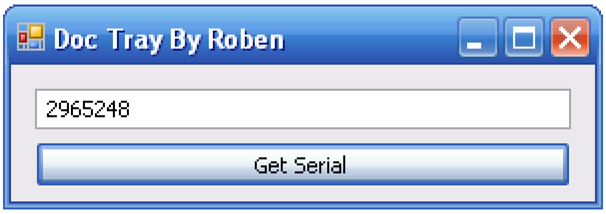
– فيصبح كود الكيجن كالتالي :

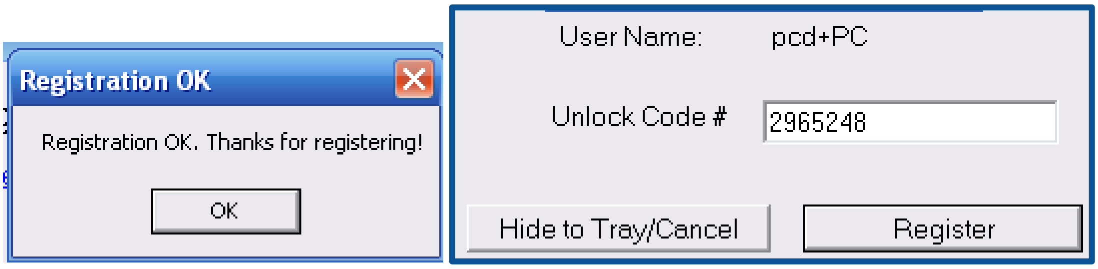
```
Sub Button1Click(sender As Object, e As EventArgs)

    Dim regdata As String
    Dim syscode3 As Integer
    Dim num2 As Integer
        regdata = Environment.UserName
        If (Strings.Len(regdata) <= 3) Then
            regdata = (Environment.MachineName).ToLower
        End If
        If (Strings.Len(regdata) <= 3) Then
            regdata = (regdata & "+" & Environment.UserName)
        End If
        If (Strings.Len(regdata) <= 3) Then
           regdata = (regdata & "+" & regdata)
        End If
        If (syscode3 = 0) Then
            num2 = Strings.Len(regdata)
            syscode3 = 0
            Dim num5 As Integer = num2
            Dim i As Integer = 1
            Do While (i <= num5)
                Dim num3 As Integer = Strings.Asc(Strings.Mid(regdata, i, 1))
                syscode3 = (syscode3 + ((&H750 * i) * num3))
                i += 1
            Loop
        End If
        Tsn.Text = Str( syscode3 )

End Sub
```

– نجرب الكيجن :





– مبارك!! ها هو يعمل بنجاح كما تشاهد.

الباب السادس: تحليل تطبيقات الجوال

كسر تطبيقات الجوال [1]

S60 Reversing (1)

1. مقدمة

يعتبر عالم كسر تطبيقات الجوال جديد نسبياً مقارنة بكسر برامج الكمبيوتر، لذلك ستلاحظ اختلافاً بين الأدوات المستخدمة لكسر برنامج كمبيوتر مع الأدوات المستخدمة لكسر تطبيق الجوال.

وللبدء في هذا الموضوع ننصحك بقراءة هذا المقال، وهو من أحسن ما كتب في كسر تطبيقات الجيل الثاني من الجوال.

أيضا نحب أن ننوه إلى أن هناك فرقاً كبيراً بين كسر تطبيقات الجيل الثاني وبين كسر تطبيقات الجيل الثالث والـ Pocket PC . وستلاحظ ذلك عند الوصول إلى هذه الجزئية في الكتاب إن شاء الله. أما الآن فنحن سنتعامل مع الجيل الثاني.

- برنامج IDA v5.2.0.908 ويمكنك تحميله من هنا
- برنامج Ultra RscEditor تجده مع المرفقات
- برنامج UnMakeSIS v0.21 تجده مع المرفقات

RingTone Studio v1.41, برنامج خاص بالتعديل على نغمات الـ Mid الموجودة في الجهاز ووضع مؤثرات عليه

وسوف يكون سير الشرح على هذا الترتيب :

1) تحسين قدرات برنامج IDA وضبط إعداداته لتتلاءم مع معالجات ARM وهي الخاصة بالجوالات.

2) استغلال المعلومات التي سنحصل عليها من ملف الريسورس الموجود مع البرنامج لتحديد المكان الصحيح للهجوم عليه.

3) استعمال برنامج IDA لتحليل البرنامج الضحية.

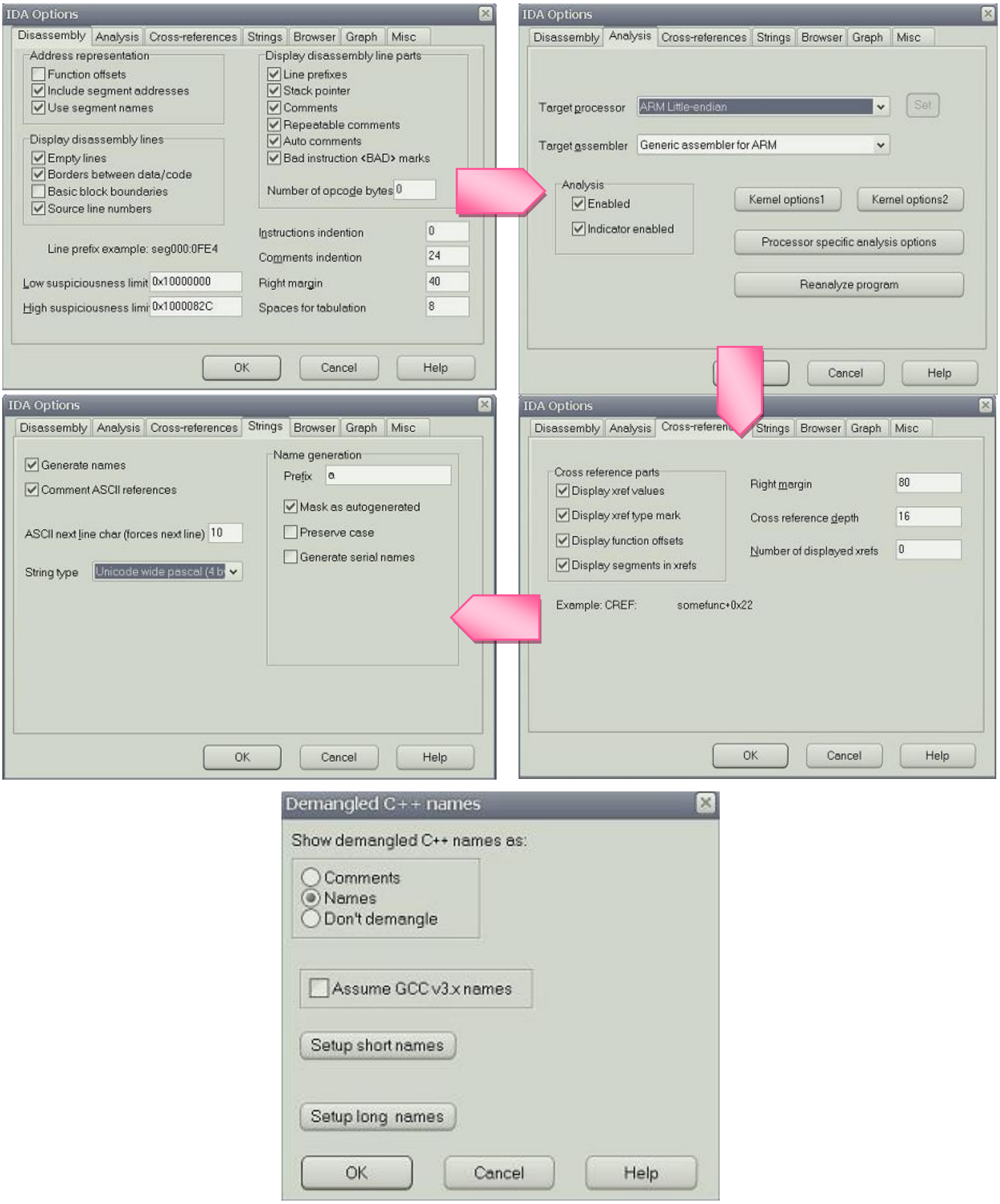
4) استعمال أي محرر هكس للتعديل على البرنامج وحفظ التغيرات.

2. الخطوة الأولى : تحسين قدرات IDA PRO

- أولاً عليك تحميل ملفات الـ IDS من هنا .

ثم استبدال الــ IDS الموجود في برنامج IDA ، وحتى لا يحدث خطأ اقرأ ملف الـــ readme.txt الموجود معهم لتتعرف على المكان الصحيح الذي يجب وضعهم فيه مع البرنامج.

- ثانياً سنقوم بضبط إعدادات برنامج IDA كما يلي :



3. الخطوة الثانية : استكشاف البرنامج

سنقوم أولاً بمعرفة ماهية الـــ Limitations الموجودة في البرنامج؟

هي حماية زمنية، عند فتح البرنامج تخبرك بأن البرنامج تجريبي لمدة 3 أيام ثم تنتهي المدة التجريبية، وتظهر لنا رسالة مزعجة تطالب بتسجيل البرنامج، ولا يمكن تشغيل البرنامج بعد ذلك إلا بعد التسجيل :



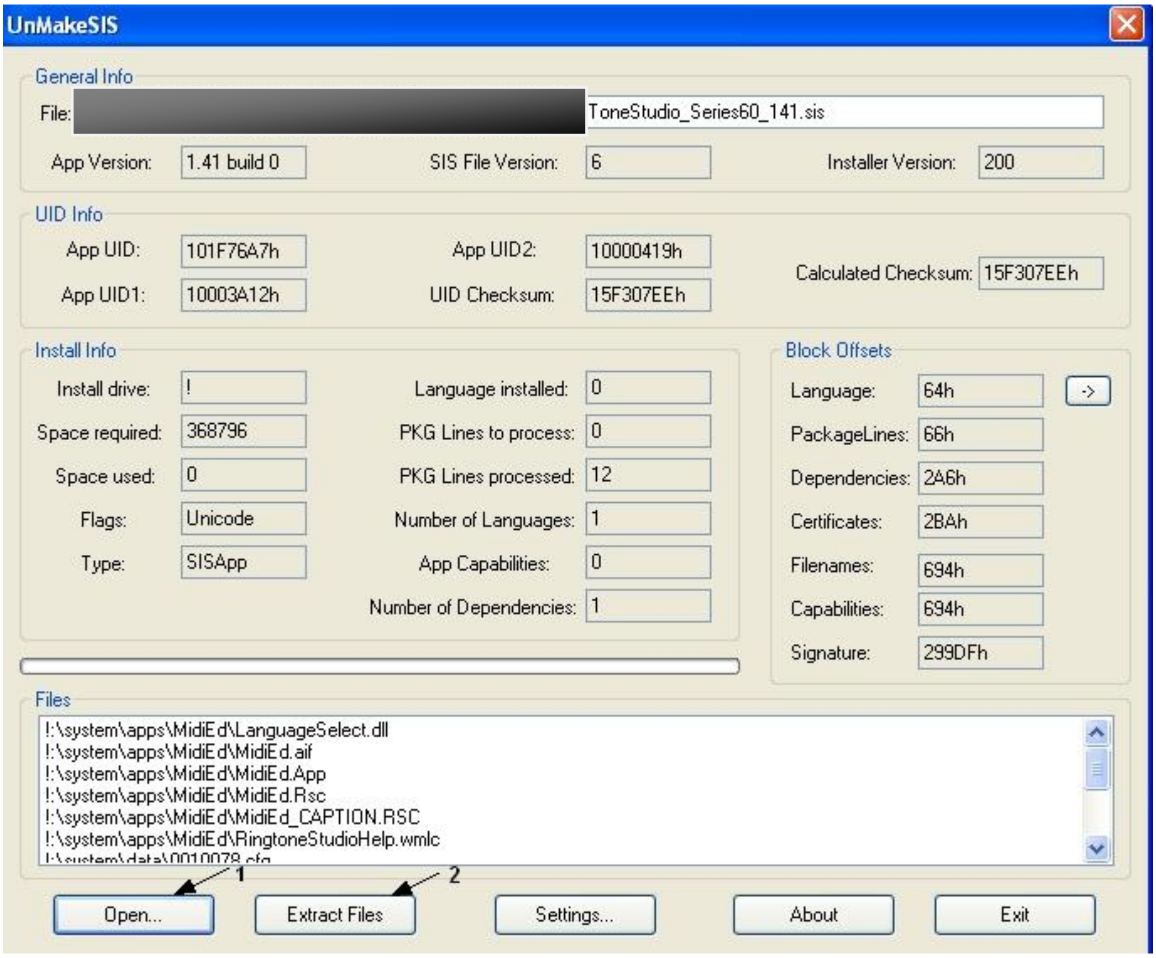
أيضاً يوجد حماية أخرى على البرنامج عند عمل Save حيث أنه لا يقوم بحفظ أي ملف جديد إلا في النسخة المسجلة.

الآن سوف نقوم باختيار Register application وسنحاول تسجيل البرنامج بأي رقم لكي نحصل على رسالة الخطأ، والتي من خلالها سيمكننا مهاجمة البرنامج :



ستلاحظ أنه قد أظهر رسالة صوتية ثم عاد إلى نفس النافذة المزعجة السابقة.

نقوم الآن بالعودة إلى الجهاز ومحاولة فك ملف الـــ SIS الخاص بالبرنامج الضحية، وسوف نستخدم لهذه المهمة برنامج UnMakeSIS :

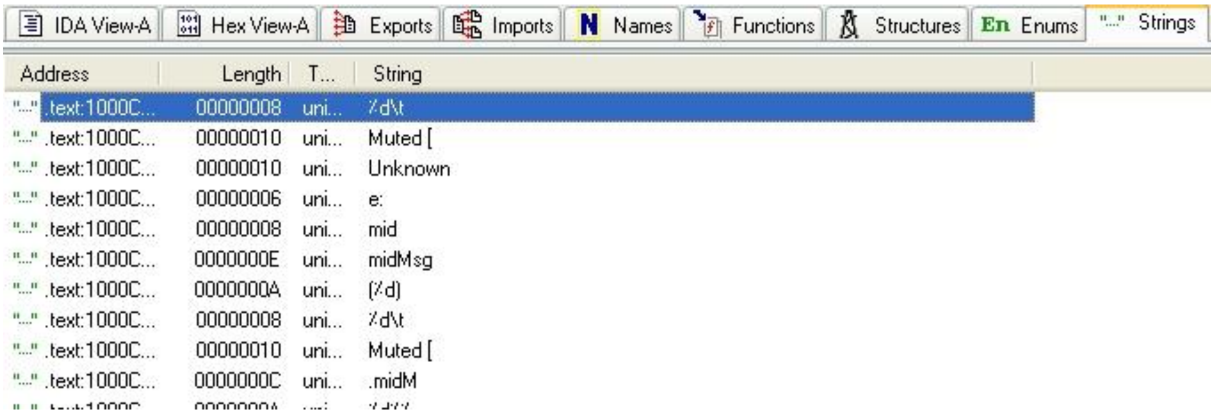


ستلاحظ أنه يتم فك ملفات البرنامج على 4 مجلدات، سندخل إلى المجلد apps ونختار الملف المنتهي بـــ .app لأن هذا الملف هو ملف تشغيل البرنامج داخل الجوال :



وسنجد الملف باسم MidiEd.App .

حسنا سنبدأ الآن تشغيل البرنامج في IDA ومحاولة البحث عن أي شيء هام في الـــ String الخاصة بالبرنامج :

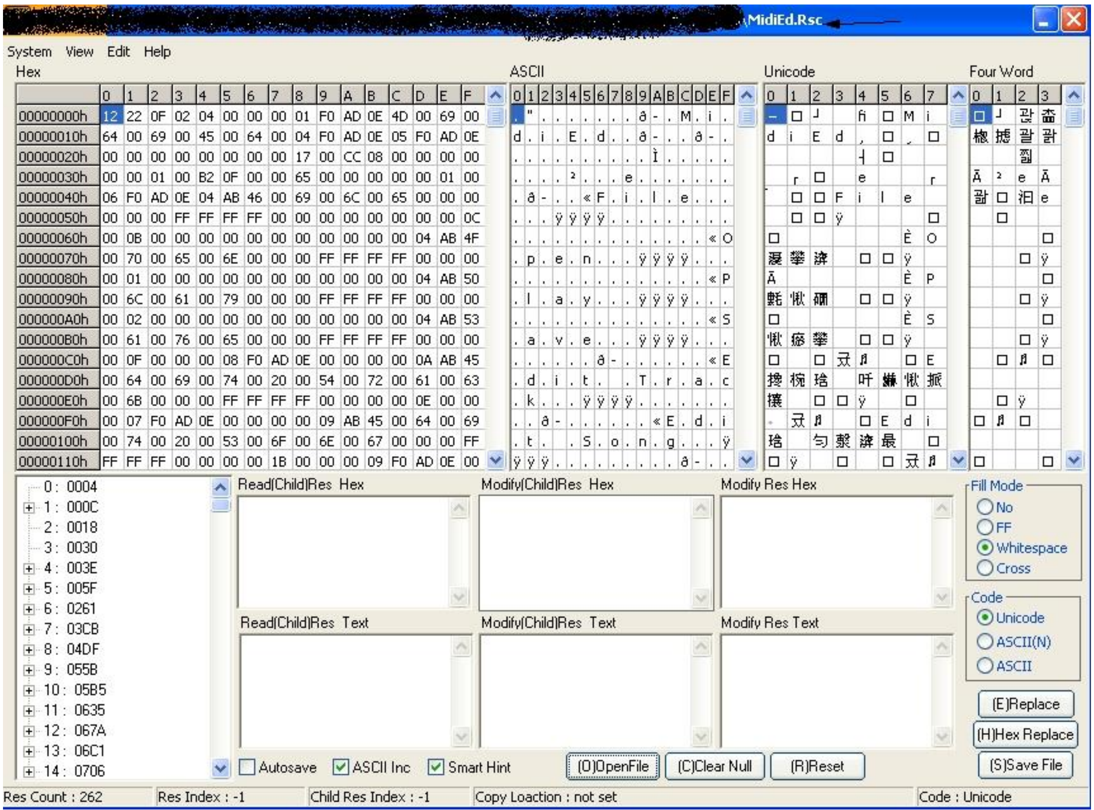


للأسف لا يوجد أي شيء هام هنا يمكننا من خلاله الوصول إلى مكان إزالة هذه النافذة المزعجة!!

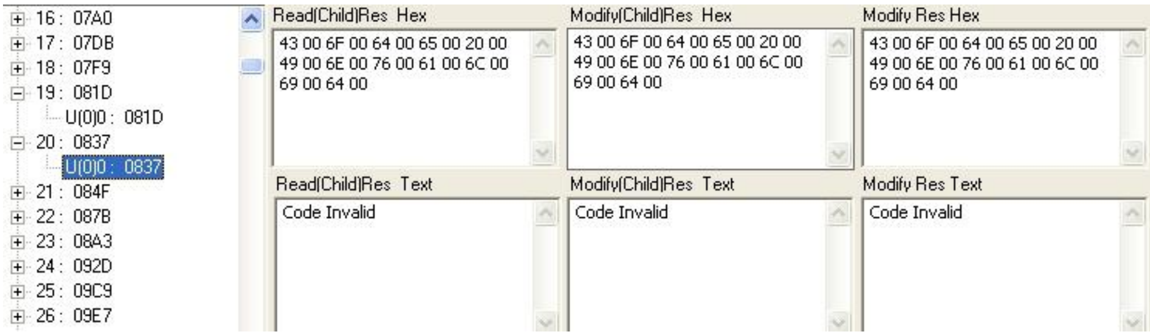
ماذا سنفعل؟ كيف سنصل إلى مكان التحقق من التسجيل وعرض الرسالة المزعجة بعد انتهاء المدة التجريبية؟

ستلاحظ بجوار ملف البرنامج ملفات أخرى تنتهي بـــ .rsc ومن اسمها يتبين أنها ملفات الريسورس الخاصة بالبرنامج، إذن فالبرنامج يقوم باستيراد أي رسائل تظهر فيه من خلال هذه الملفات، ولا يتم حفظ الرسائل داخل البرنامج نفسه، وهذه الحماية موجودة في الكثير من برامج الجوال.

سنقوم بفتح ملف MidiEd.Rsc بواسطة برنامج خاص للتعامل مع هذه النوعية، وهو UltraRscEditor :



أثناء بحثنا عن النصوص العامة سنلاحظ سترينغ هامة جداً وهي رقم 20 حيث أنها الخاصة برسالة الخطأ Code Invalid :



أيضاً ستلاحظ أن السترينغ رقم 19 يحتوي على Code Accepted لذلك سنقوم بمهاجمة هذا الملكان، ولكن كيف؟



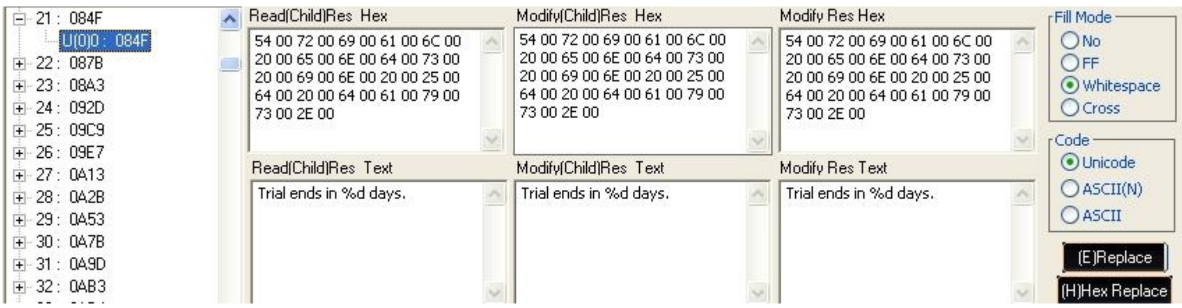
هناك دالة هامة جداً يتم استخدمها في برامج الجوال لجلب النصوص من ملف الريسورس وهي الدالة ReadResourceAsDes .

هذه الدالة دائماً يسبقها عدد بالهكس، وهو عنوان السترينغ في ملف الريسورس، والذي ستتم قراءته بواسطة هذه الدالة :

```
LDR     R2, =0xEADF020
BL      ReadResourceAsDes__CCCoeEnvRTDes16i
```

ستلاحظ مثلاً في الصورة السابقة أن الدالة يسبقها العدد EADF020 أي أن هذا عنوان لشيء ما في الريسورس سيتم استدعاؤه. ومن خلال التجربة خلصت إلى نتيجة يمكن استغلالها في كل البرامج التي تستخدم هذه الطريقة وهو أن العدد EADF ثابت ويليه ثلاثة أرقام متغيرة من دالة إلى أخرى، هذه الأرقام هي ترتيب السترينغ داخل ملف الريسورس.

يعني مثلاً مع هذا البرنامج وجدنا –عندما شاهدنا ملف الريسورس الخاص به– أن النصوص مرتبة فيه من 0 إلى 261 فلو أخذنا العدد السابق EADF020 سيصبح رقم السترينغ المطلوب المفروض هو رقم 20 لا تنس أن العدد بالهكس يحول إلى Decimal ويصبح 32، لكن بما أن العد بدأ من الصفر فسيكون هذا الرقم EADF020 هو عنوان السترينغ رقم 33 داخل ملف الريسورس :

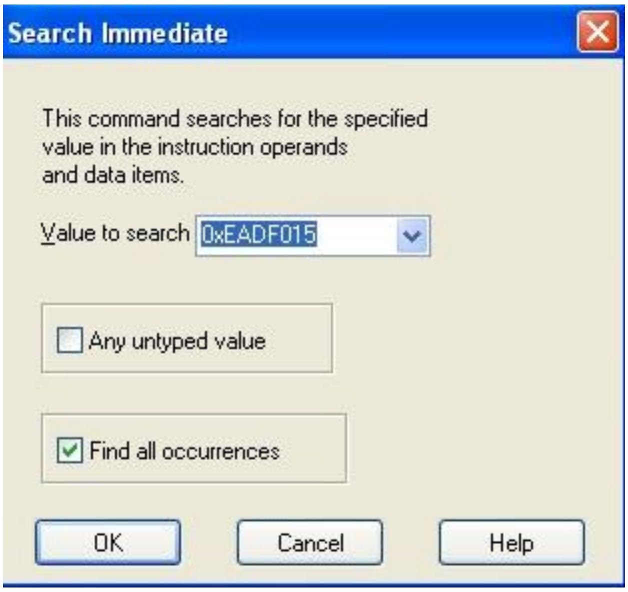


ملحوظة هامة العدد EADF متغير من برنامج لآخر؛ لذلك فنحن سنأخذ فقط آخر رقمين عند المحاولة مع برنامج آخر.

الآن يمكننا معرفة الرقم الذي سيسبق دالة ReadResourceAsDes عندما يريد البرنامج جلب رسالة الخطأ Code Invalide .

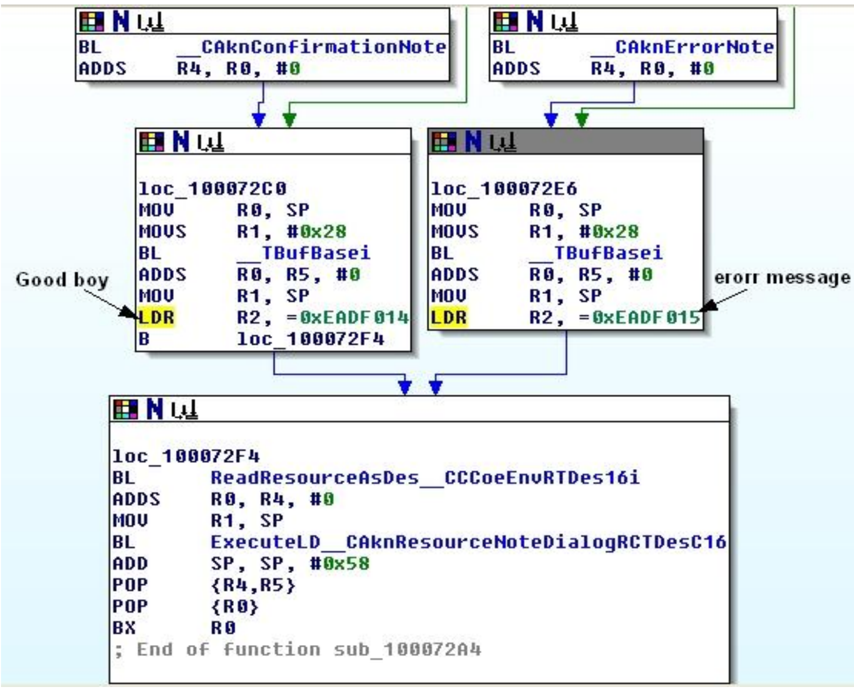
رقم السترينغ في ملف الريسورس هو 20 أضف إليه 1 سيصبح 21 حوله إلى هكس سيصبح 15 إذن عنوان السترينغ الذي سيستدعيه البرنامج سيكون EADF015 .

نعود الآن إلى برنامج الـ IDA ثم Search ثم Immediate Value ونكتب xEADF0150 :



4. الخطوة الثالثة : كسر البرنامج

تم الوصول إلى روتين التحقق من السريال، ثم إظهار رسالة الصواب أو رسالة الخطأ :

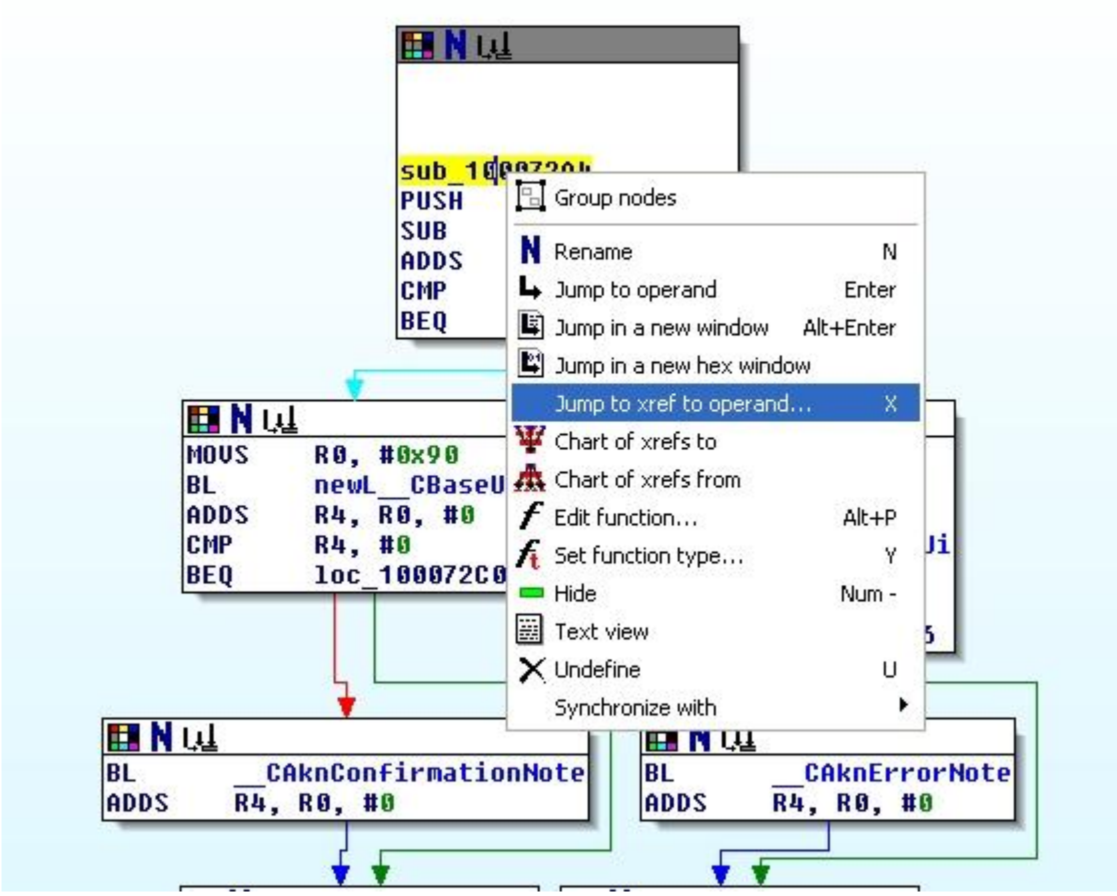


ربما يقول البعض: لماذا لا نعكس القفزة الموجودة في بداية الروتين وبعد ذلك ندخل أي سريال لتسجيل البرنامج؟

هذه الطريقة خاطئة على الرغم من أنه بعد إدخال أي سريال سوف يعرض لك رسالة Code Accepted لكن سيظل البرنامج غير مسجل، ولن تستطيع حتى فتحه بعد انتهاء المدة التجريبية.

الحل الأمثل هو تتبع القفزة التي تؤدي بنا إلى هذا الروتين ومحاولة عكسها حتى لا يعرض لنا البرنامج هذه النافذة المزعجة بعد انتهاء مدة التجريب.

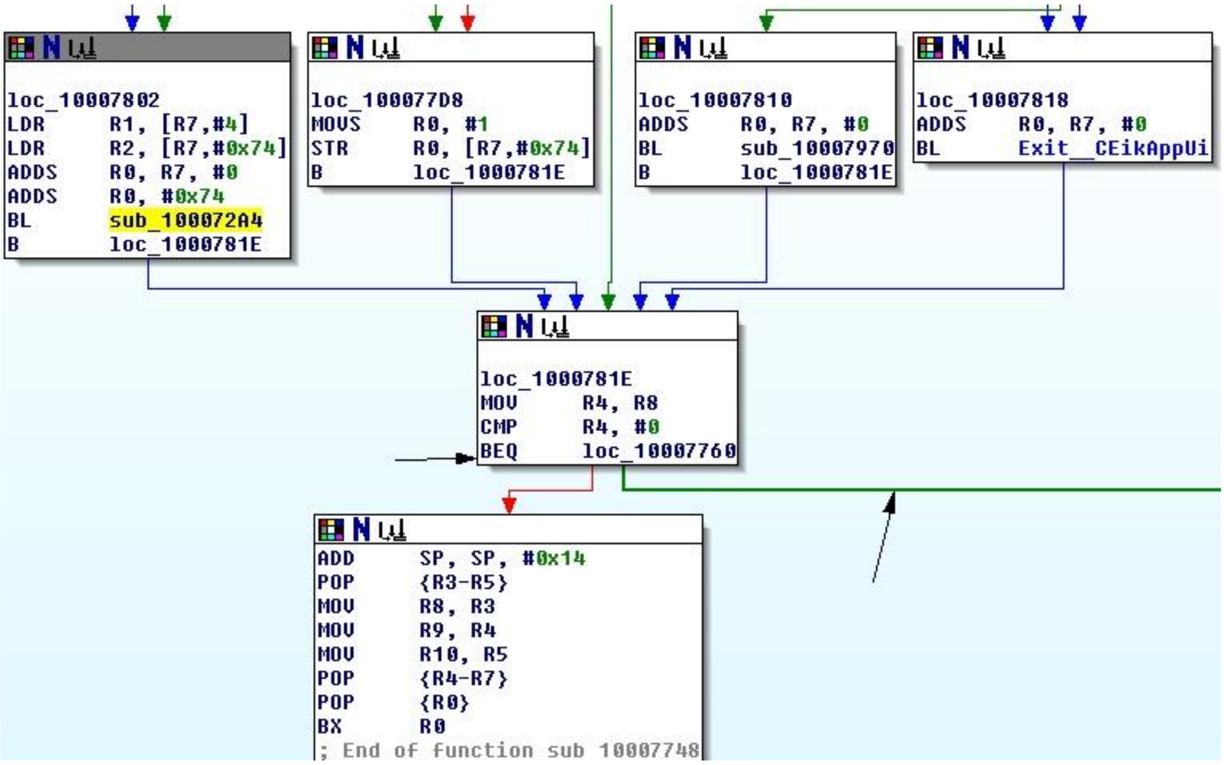
اذهب إلى بداية الروتين واضغط على X من لوحة المفاتيح :



ستظهر لنا نافذة تحتوي على مكانين يتم استدعاء هذا الروتين عندهما :

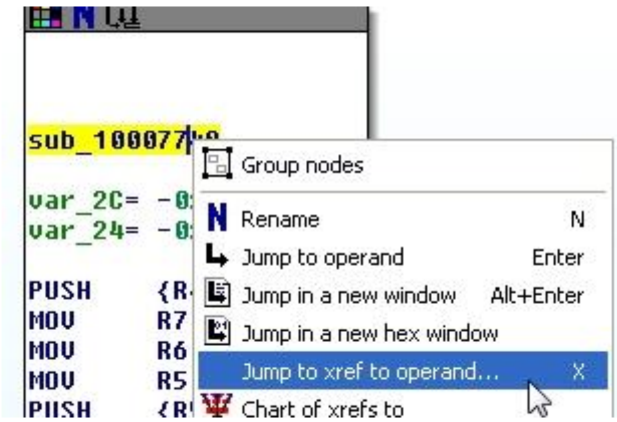


ندخل إلى الاستدعاء الثاني؛ لأن الأول عديم الفائدة وليس هو المكان الصحيح الذي يختبر فيه البرنامج السريال والــ Trial Time Protection :

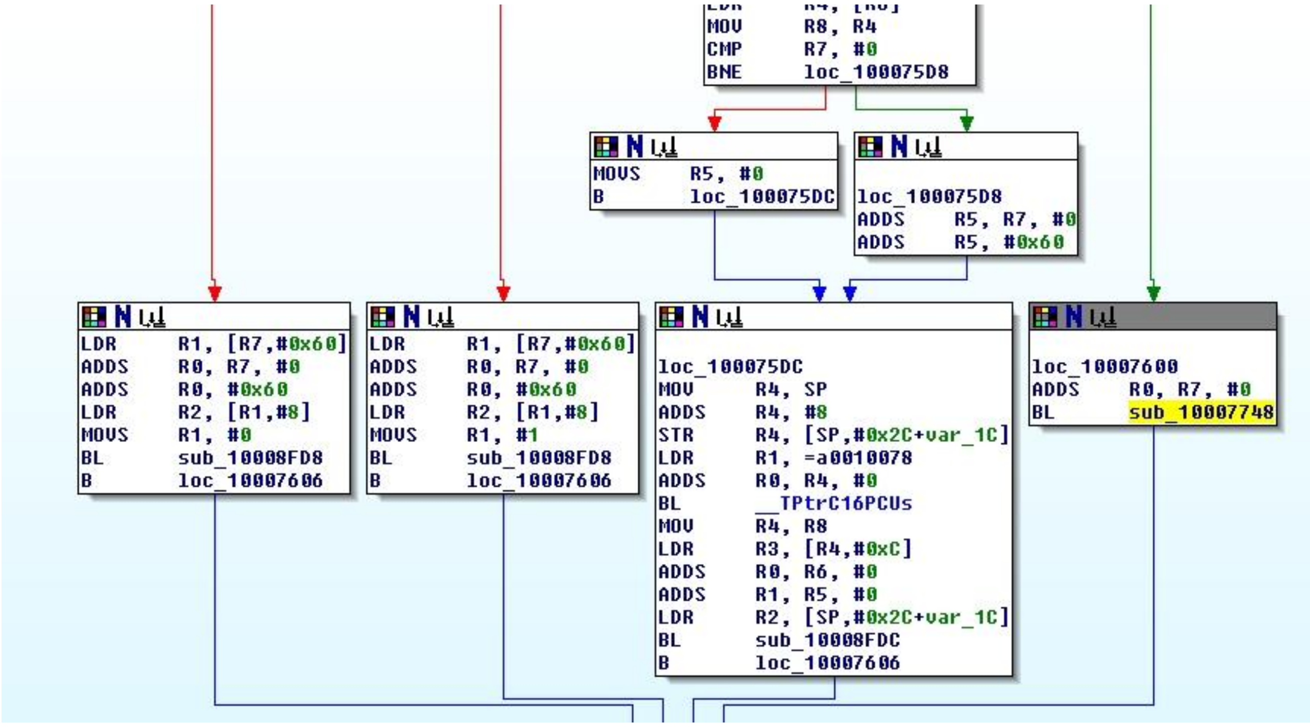


إذن يتم استدعاء الروتين السابق من عند BL sub_100072A4 وعند محاولتك لإدخال السريال الخطأ في مرحلة استكشاف البرنامج لاحظنا أنه بعد سماعك للرسالة الصوتية – والتي تعني أن السريال خطأ– يقوم البرنامج بإعادة عرض النافذة المزعجة التي تطالبك بالتسجيل مرة أخرى، وهذا يعني أن هنالك حلقة مسؤولة عن ذلك، ولا يتم الخروج منها إلا بعد إدخال السريال الصحيح، وستلاحظ ذلك في الصورة السابقة عند القفزة BEQ loc_10007760 والخط الأخضر هو سير البرنامج إذا تمت القفزة، وكما ترى فإن هذا الخط يأخذنا إلى بداية الروتين مرة أخرى أي أنه حلقة.

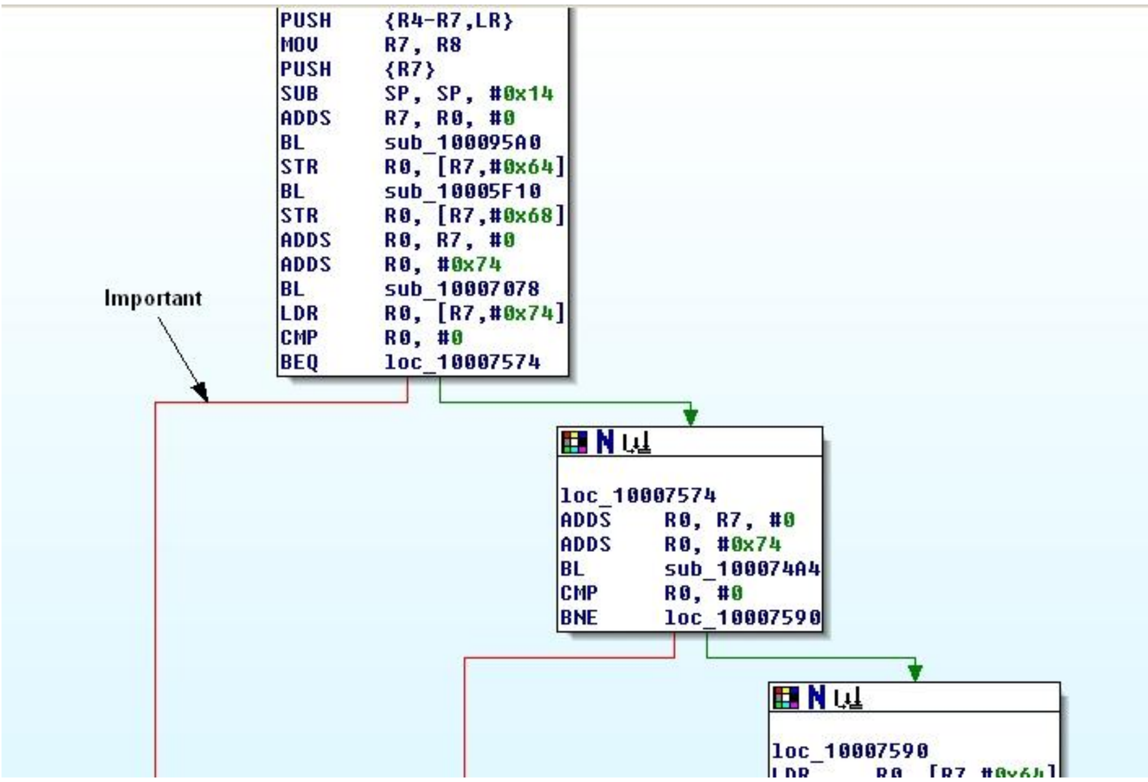
نستنتج من ذلك: أن هذا الروتين هو الذي يحتوي على النافذة المزعجة التي تطالب بالتسجيل، والتي تظهر بعد انتهاء المدة التجريبية للبرنامج، لذلك سنبحث عن كل الاستدعاءات التي تؤدي بنا إلى هذا الروتين كما فعلنا مع الروتين السابق :



ستلاحظ أنه استدعاء واحد، لندخل إليه :



نصعد للأعلى إلى بداية هذا الروتين، سنجد قفزة هامة جداً إذا تمت ستأخذنا بعيداً عن النافذة المزعجة وعن الـــ Trial Time :



الآن نريد عكس القفزة BEQ loc_10007574 لذلك نأخذ الأوفست الخاص بها :

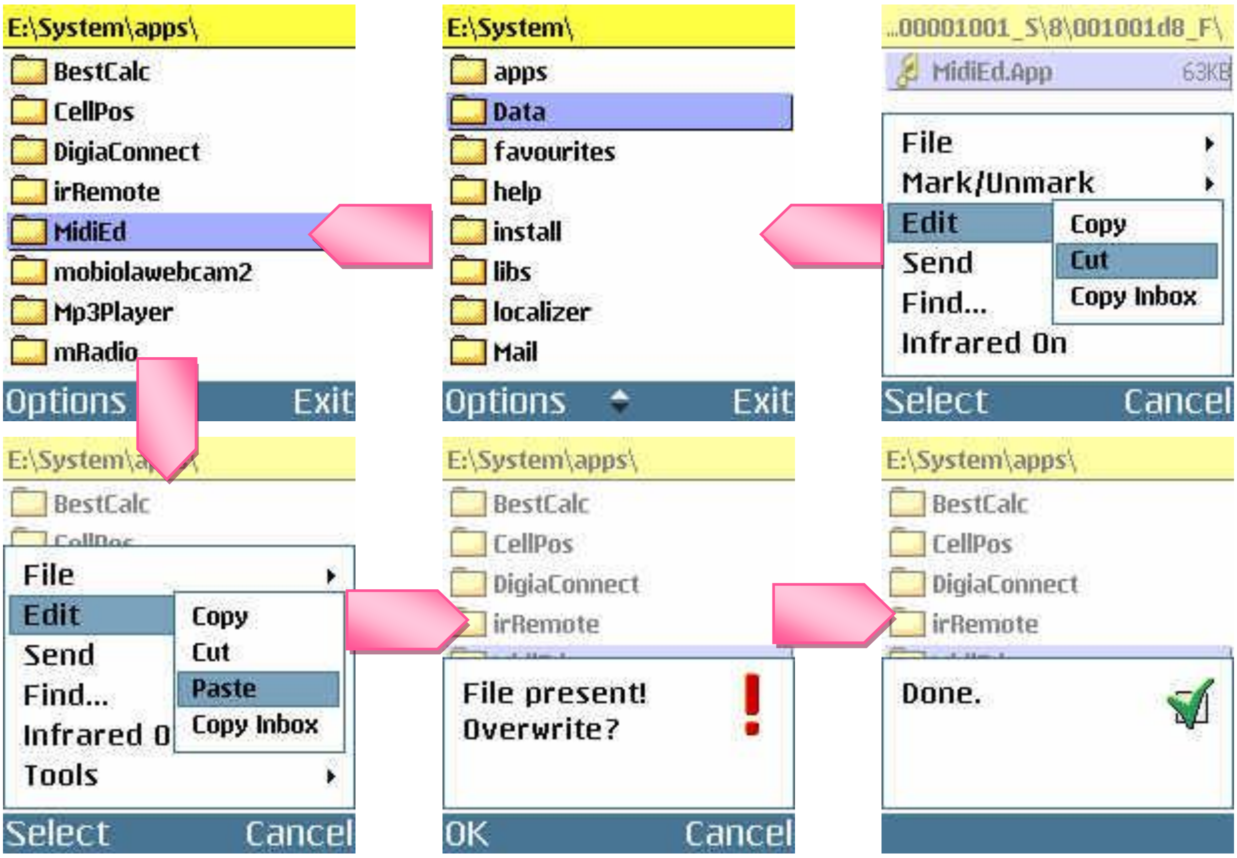


الأوفست هو DE75 سنقوم بفتح أي محرر هكس، وسنذهب إلى هذا الأوفست، سنجد أن الـــ BEQ تساوي D0 نعكس الـــ D0 إلى D1 تصبح BNE أي أننا عكسنا القفزة :



نقوم بحفظ الملف بنفس الاسم MidiEd.App ثم نرسل هذا الملف إلى الجوال.

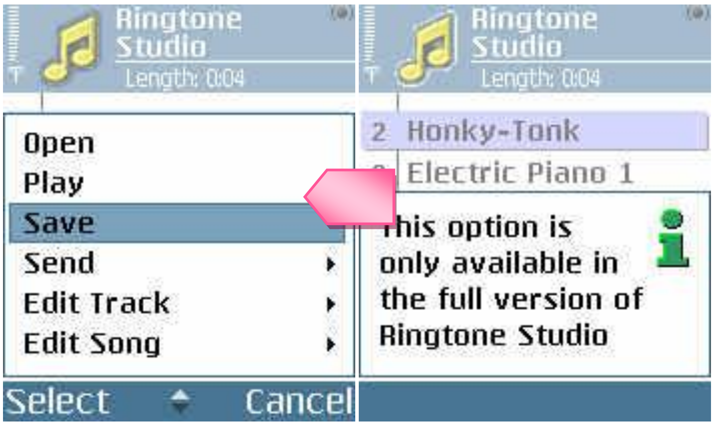
بواسطة أي برنامج مثل الـــ SeleQ نقوم بأخذ الملف من الرسائل ثم عمل Cut له و Paste في مجلد البرنامج :



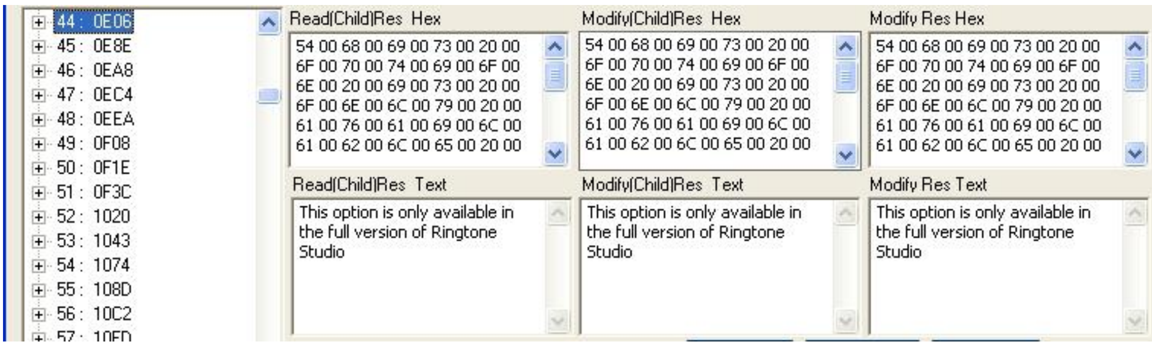
الآن سنقوم بتجريب البرنامج. مبارك البرنامج يعمل بدون أي رسائل و حتى بعد انتهاء المدة التجريبية :



لكن للأسف مازال البرنامج غير مسجل!! جرب أن تدخل إلى ملف MIDI وحاول التعديل عليه، وجرب عمل Save :



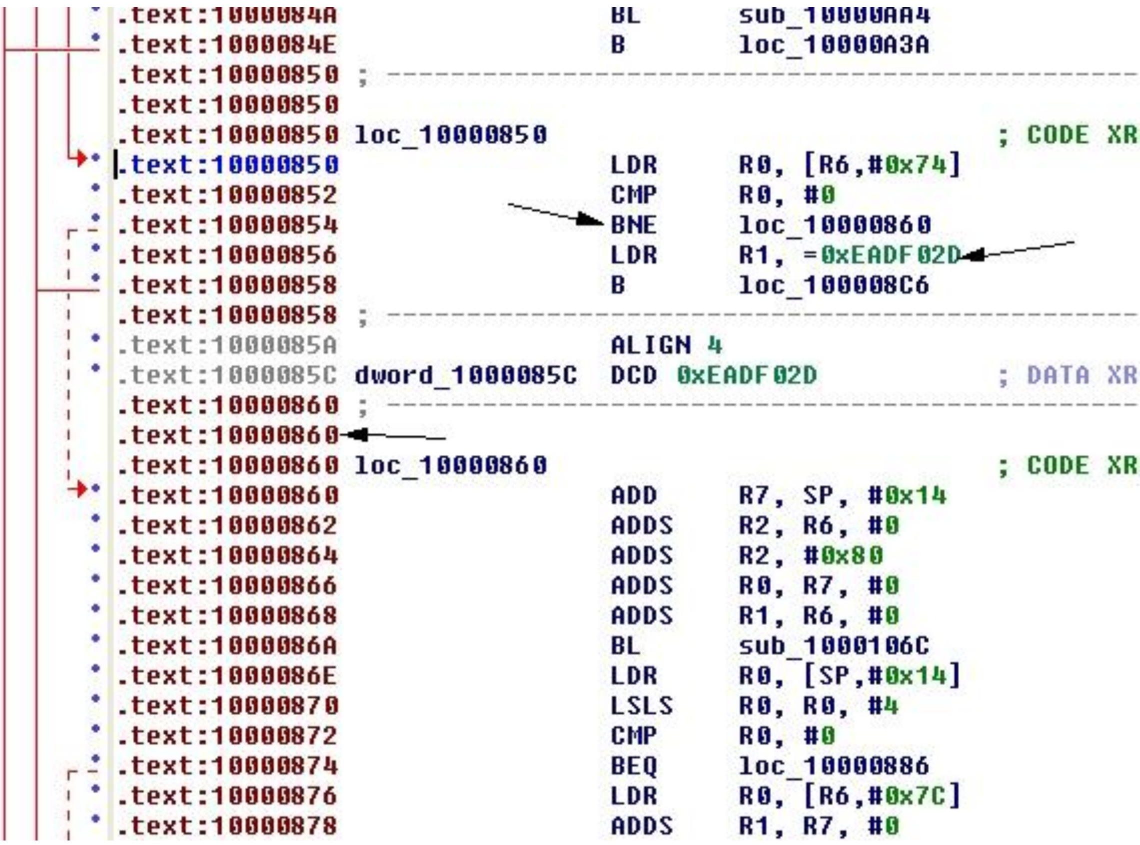
لكن اعتقد الخطوات أصبحت واضحة، وهي الذهاب إلى ملف الريسورس والبحث عن السترينغ This option is only available وبعد ذلك البحث عن الـ Immediate Value كما فعلنا سابقاً وعكس القفزة التي تؤدي إلى هذه الرسالة :



إذن هي رقم 44 نضيف واحداً تصبح 45 نحول إلى هكس تصبح D2 إذن الـ Immediate Value تساوي EADF02D :



سنجد استدعائين، وبعد التجريب تبين أن الاستدعاء الثاني هو المسؤول عن هذه الرسالة فلندخل إليه :



ماذا لو عكسنا القفزة الموجودة عند العنوان 10000854 ؟ ربما ستأخذنا إلى رويتي جديد؟ ربما يكون السؤال عن إجرائيات حفظ الملف بدل من عرض النافذة المزعجة.

لنأخذ الأوفست لهذه القفزة

```
000008D0   10000854: .text:10000854
```

نعكس الـــ BNE إلى BEQ عن طريق عكس الـــ D1 إلى D0 :

```
000008B8 00D1 FCE0 301C 0AF0 0DF9 F8E0 301C 00F0 2BF9 F4E0 706F 0028 04D0 0
000008DC 05AF 321C 8032 381C 311C 00F0 FFFB 0598 0001 0028 07D0 F06F 391C 0
00000900 1FFA 8724 A400 6C44 9594 05A9 201C 0AF0 5BFE 9598 0AF0 5EFE 8FAA 6
```

احفظ الملف بنفس الاسم وانقله إلى الجوال كما فعلنا سابقاً. والآن جرب حفظ أي ملف معدل.

مبارك تم تخطي النافذة المزعجة، وعرض لك نافذة إدخال الاسم الذي تود حفظ الملف الجديد به :



احفظه مثلا باسم At4re ثم اضغط على OK سيتم حفظ الملف في مجلد الصوتيات في الـــ Phone Memory :



وبذلك نكون انتهينا بحمد الله من كسر هذا البرنامج.

كسر تطبيقات الجوال [2]

S60 Reversing (2)

I. مقدمة حول المعالج ونظام التشغيل

في هذا الدرس سنتعامل مع نوع جديد من المعالجات Processors وهو معالج ARM ونظام تشغيل آخر، وهو ما يسمى Symbian Operating System ولن نخوض في شرح حول معالج ARM أو نظام التشغيل وتفاصيلهما؛ لأن الدرس لا يتسع لذلك، لذا سنحاول سرد بعض المعلومات حول التعامل مع تطبيقات الجيل الثالث في كل ما يتعلق بالهندسة العكسية فقط، ولمزيد من التفاصيل حول المعالج ونظام التشغيل سنسرد بعض الروابط المفيدة في آخر الدرس. لابد من توفر بعض الأدوات اللازمة للعمل :

- IDA v5.2.0.908 : يمكنك تحميله من هنا .
- SisWare : يستخدم لاستخراج ملفات البرنامج المضغوطة في ملف sis .
- Petran : لفك ضغط الملف التنفيذي للبرنامج وإعادة ضغطه بعد الكسر.
- WinHex : محرر هكس لتعديل الـ Opcodes للتعليمات التي نريد تغييرها.
- X-plore v1.30 : متصفح ملفات للجوال مهم جداً بعد اختراق حماية الجوال.

- expenses-s60-3-v-2-0.sis
- smartcom_s60_3rd_trail.sis

ولا تنس أيضاً برنامجاً للاتصال بين الجوال والكمبيوتر؛ مثل: Nokia PC Suite

2. اختراق حماية الجوال وفتح مجلدات النظام

الشيء الجديد الذي أضيف في نظام التشغيل للجيل الثالث هو نوع من الحماية مدعوم بإضافة شهادة Certificate خاصة لكل برنامج ولكل جوال، وإخفاء ملفات النظام في مجلدات Sys & Private حتى لا يتم العبث بها، الأمر الذي لم نعهده في جوالات الجيل الثاني من سمبيان S60 ، وغالباً عند تثبيت تطبيق S60v3 تم كسره App Cracked ستجد رسالة خطأ في الجوال تفيد أن شهادة التطبيق منتهية أو أنه من مورد غير موثوق به، في بعض الأحيان يتم تعديل تاريخ الهاتف عاماً أو عامين إلى الوراء وتحل المشكلة، ولكن في كثير من الأحيان لا يتم ذلك، وهذا يعتبر معضلة بالنسبة للكراكر؛ لأن معظم البرامج التي يقومون بكسرها لا تنفع بدون شهادة أو اختراق للحماية؛ لذلك فقد تم عمل طرق كثيرة لاختراق حماية الجوال، بعضها بسيط والبعض الآخر معقد جداً، منها التي تعمل بنجاح، ومنها الطرق التي تعمل جزئياً، كما تم برمجة بعض التطبيقات التي تقوم بفك الحماية أوتوماتيكياً، لذلك سنترك لك البحث عن الطريقة أو البرنامج الذي يلائمك، وننصح ببرنامج BiNPDA Security Manager v1.1 لأننا جربنا الكثير من الطرق فوجدنا أن هذا البرنامج هو أفضلها. كما يمكن الحصول على شهادة خاصة بجوالك من بعض المواقع غير موقع Symbiansigned.com وأغلبها مواقع صينية، بعد أن تجد أحد هذه المواقع يتم تقديم طلب شهادة للـــ IMEI الخاص بجوالك، وتحصل عليها بعد يوم أو يومين، عندها تستطيع دمجها مع البرامج التي لا تحوي شهادة وذلك باستعمال أحد برامج الـــ Signing .

3. طريقة العمل في هذا الدرس

- الخطوة الأولى : استخراج الملف التنفيذي للتطبيق من ملف sis بواسطة SISWare وفك ضغطه بواسطة Petran.
- الخطوة الثانية : تحميل الملف التنفيذي في Pro Advanced v5.2 IDA والبدء في التحليل.
- الخطوة الثالثة : تعديل الـــ Opcodes للتعليمات اللازم تغييرها باستخدام WinHex أو أي محرر هكس آخر.
- الخطوة الرابعة : إعادة ضغط الملف التنفيذي بواسطة Petran واستبدال الملف المثبت في الجوال.

تم عمل هذا الدرس بالتجربة على جوال Nokia N73 وبعد اختراق الحماية.

لنبدأ على بركة الله.

4. كسر تطبيقات الجوال

4.1. التطبيق الأول

اسم التطبيق : Expenses v2.0.

اسم المورد : SymbianGuru.

رابط الضحية : www.symbianguru.com.

بعد تثبيت التطبيق في الجوال وفتحه نجد أن مدته التجريبية 10 أيام، كما تلاحظ ذلك في الصورة :



وفي قائمة Options تجد خيار Register وهذا ما سنستفيد منه لكسر التطبيق، إذن لننتقل إلى الخطوات.

قم بتنفيذ الخطوتين الأولى والثانية وانتظر حتى يتم الانتهاء من تحميل الملف في IDA وعندها افتح نافذة Imports وقم بالبحث عن TDesCCompareERKS لكن لماذا ؟

في كثير من التطبيقات يتم استخدام هذه الدالة لعمل المقارنة بين بيانات Data معينة، والتي قد يكون منها اختبار صحة السريال المدخل أو أن البرنامج مسجل أو تجريبي وما إلى هنالك.

دعنا نرى تعريف هذه الدالة في مكتبة Symbian OS 9.1 Developer :

class TDesC8;	
Members:	
Compare()	
Prototype :	
IMPORT_C TInt Compare(const TDesC8 &aDes) const;	
Parameters	
const TDesC8 &aDes	**The 8-bit non-modifable descriptor whose data is to be compared with this descriptor's data,** تتم المقارنة بين **8–bit non-modifiable Descriptor's Data** و **Specified Descriptor's Data** بايتاً مع

	بايت ،
Return value	
Tint	Positive, if this descriptor is greater than the specified descriptor. Negative, if this descriptor is less than the specified descriptor. Zero, if both descriptors have the same length and their contents are the same.

وتتساوى البيانات عندما يكون طولهما ومحتواهما متساويين، وفي هذه الحالة تكون القيمة المرجعة أو ما يسمى (Return Value) للمسجل (Register) هي الصفر، وما عدا ذلك تكون قيمة المسجل إما موجبة أو سالبة :

000A9290	1821	TDesCAllocLCEv_0	euser
000A9204	1728	TDesCAllocLEv	euser
000A90...	1817	TDesCAllocLEv_0	euser
000A9268	1718	TDesCAtCEi	euser
000A91...	1806	TDesCAtCEi_0	euser
000A9174	1733	TDesCCompareERKS	euser
000A9218	1822	TDesCCompareERKS_0	euser
000A9258	1818	TDesCLocateETChar	euser
000A9150	1820	TDesCMatchFERKS	euser
000A92...	2108	TDesCMidEi	euser

ولكن هذه الدالة تستخدم لأغراض أخرى كثيرة، لذلك يجب التركيز على المكان الصحيح الذي يخص دالة التحقق من التسجيل، وغالباً ما تكون متبوعة بتعليمتي MOVNE & MOVEQ حيث يتم نقل القيمة 1 إلى المسجل في حالة كان السريال المدخل صحيحاً، أو 0 عندما يكون خطأ، قم بالضغط مرتين على ما هو موضح في الصورة وستجد نفسك هنا :

```
.idata:000A9170 ?? ?? ?? ??          IMPORT __imp_RArrayBaseInsertIsqEPKvPFiSSEi
.idata:000A9170                                          ; CODE XREF: RArrayBaseInsertIsqEPKvPFiSSEi↑j
.idata:000A9170                                          ; DATA XREF: .text:off_2DCC4↑o
.idata:000A9174 ?? ?? ?? ??          IMPORT __imp_TDesCCompareERKS   ; CODE XREF: TDesCCompareERKS↑j
.idata:000A9174                                          ; DATA XREF: .text:off_2DCF4↑o
.idata:000A9178 ?? ?? ?? ??          IMPORT __imp_RArrayBaseBinarySearchEPKvRiPFiSSE
```

ضع مؤشر الفأرة على imp_TDesCCompareERKS__ واضغط على حرف "X " للذهاب إلى هنا :

xrefs to __imp_TDesCCompareERKS

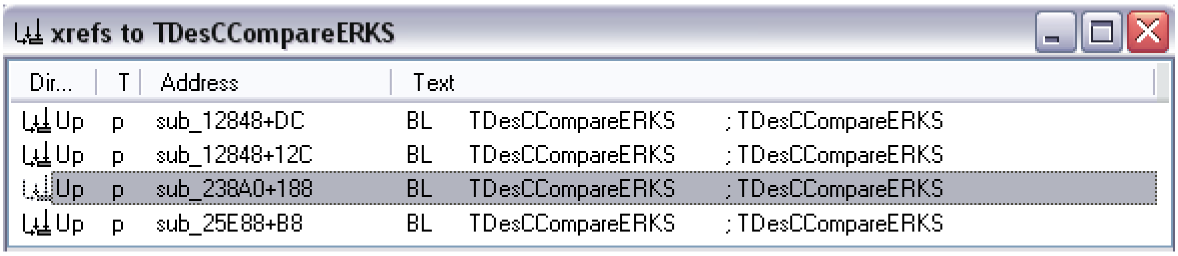
Dir...	T	Address	Text
Up	j	TDesCCompareERKS	LDR PC, =__imp_TDesCCompareERKS; off_2DCF4
Up	o	.text:off_2DCF4	DCD __imp_TDesCCompareERKS

ادخل على الخيار الأول لتجد هذا :

```
.text:0002DCF0
.text:0002DCF0                 ; Attributes: thunk
.text:0002DCF0
.text:0002DCF0                 TDesCCompareERKS                        ; CODE XREF: sub_12848+DC↑p
.text:0002DCF0                                                         ; sub_12848+12C↑p ...
.text:0002DCF0 000 04 F0 1F E5         LDR     PC, =__imp_TDesCCompareERKS ; Indirect Jump
.text:0002DCF0 000             ; End of function TDesCCompareERKS
.text:0002DCF0
```

مؤشر الفأرة على TDesCCompareERKS ثم "X" لتفتح لك هذه النافذة :

اختر الـــ Reference الثالث واضغط OK *** لأنه – بعد التحليل – هو الذي يخص دالة التحقق من التسجيل ***



xrefs to TDesCCompareERKS

Dir...	T	Address	Text		
Up	p	sub_12848+DC	BL	TDesCCompareERKS	; TDesCCompareERKS
Up	p	sub_12848+12C	BL	TDesCCompareERKS	; TDesCCompareERKS
Up	p	sub_238A0+188	BL	TDesCCompareERKS	; TDesCCompareERKS
Up	p	sub_25E88+B8	BL	TDesCCompareERKS	; TDesCCompareERKS

والآن ها نحن عند أهم تعليمتين MOVNE & MOVEQ :

```
B0 28 00 EB        BL      TDesCCompareERKS
54 00 0B E5        STR     R0, [R11,#var_54]
54 30 1B E5        LDR     R3, [R11,#var_54]
00 00 53 E3        CMP     R3, #0
00 30 A0 13        MOVNE   R3, #0
01 30 A0 03        MOVEQ   R3, #1
```

كما ترى هناك مقارنة بين المسجل والقيمة 0 فإن كانت قيمة المسجل غير مساوية للصفر فستنقل له القيمة صفر، وفي الحالة الأخرى عندما تكون قيمته صفراً ينقل له الواحد، وقد عرفنا أن قيمة المسجل لا تكون صفراً إلا عندما تكون البيانات متساوية بايتاً بايتاً في الطول والمحتوى، أي أن السريال المدخل يساوي تماماً السريال الصحيح الذي ينبغي أن نسجل به البرنامج.

وهكذا يكون التعديل على التعليمة الأخيرة في الصف بتحويلها إلى MOV R3, 1 فمهما كان ناتج العمليات قبلها فإن آخر قيمة ستنقل للمسجل هي الواحد، وبالتالي يتم تسجيل البرنامج. إذن علينا تعديل بايتات التعليمة MOVEQ R3, #1 بحيث تصبح MOV R3, #1 :

البايتات الأصلية	البايتات المعدلة	عنوان الأوفست Offset
01 30 A0 03	01 30 A0 E3	1BAD8

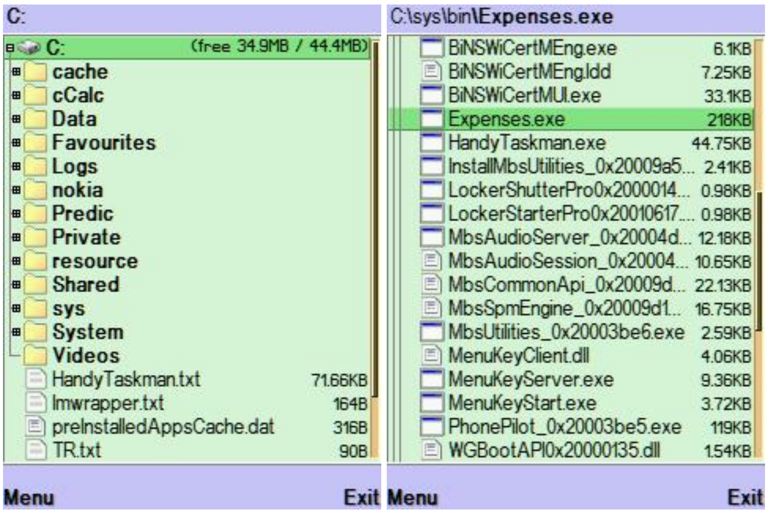
انتقل إلى الخطوة الثالثة لتعديل اللازم وهذا ما ستراه بعد التعديل :

```
B0 28 00 EB        BL       TDesCCompareERKS
54 00 0B E5        STR      R0, [R11,#var_54]
54 30 1B E5        LDR      R3, [R11,#var_54]
00 00 53 E3        CMP      R3, #0
00 30 A0 13        MOVNE    R3, #0
01 30 A0 E3        MOV      R3, #1
```

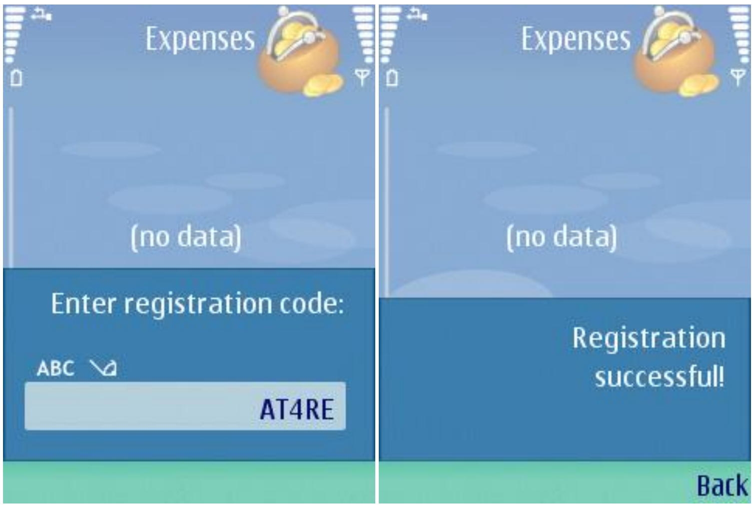
انتقل إلى الخطوة الرابعة ثم استبدل الملف التنفيذي المعدّل بالقديم (بعد كسر الحماية فقط).

أو أعد ضغط الملفات كلها بواسطة SISWare إلى ملف sis مع استبدال الملف التنفيذي وقم بدمج الملف الناتج sis بالشهادة التي لديك (هذا في حال لم تقم بفك الحماية).

هنا ستجد الملف التنفيذي للتطبيق داخل الجوال باستخدام متصفح X-plore v1.30 وهذا طبعاً لا يتم إلا بعد اختراق حماية الجوال لفتح مجلدات النظام :

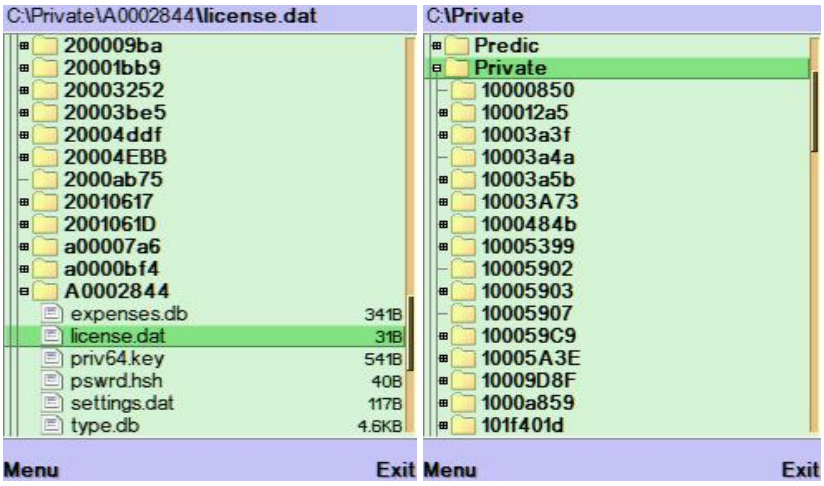


بعد استبدال الملف قم بفتح البرنامج وسجل بأي رقم أو حرف :



تلاحظ أنه تم التسجيل بنجاح وعند إعادة التشغيل يقوم البرنامج بتفعيل كافة الخصائص. مبارك!!

* ملاحظة : يتم حفظ مفتاح التسجيل في ملف License.dat داخل مجلد c:\private\A0002844



4.2. التطبيق الثاني

	اسم التطبيق : Smart Com v1.00. اسم المورد : Aims MIGITAL Technovations. رابط الصحة : www.migital.com.

لحسن الحظ فجميع برامج هذه الشركة لها نفس أسلوب الحماية، وهي سهلة جداً، ويتم فكها عن طريق البحث عن رسائل الخطأ وما شابهها؛ مثل: Trial is expired وهذا بالضبط ما سنعمله لكسر هذا التطبيق.

عند فتح التطبيق في الجوال ستجد في القائمة الرئيسية ما يلي :

عند فتح البرنامج لأول مرة Continue Trial وبعد انتهاء المدة التجريبية Trial Expired كما في الصورة :

لديك اقتراح أو سؤال حول هذه الطبعة؟ اضغط هنا



انتقل إلى الخطوتين الأولى ثم الثانية، وبعدها افتح نافذة Strings في برنامج IDA وقم بالبحث عن الكلمات التي تراها في الصورة وهي Trial Expired وهذا ما ستجده :

.text:00080...	00000032	unicode	Registration Successful.
.text:00080...	0000003E	unicode	Invalid Key! Please try again.
.text:00080...	0000001C	unicode	Licensed User
.text:00080...	00000018	unicode	Continue...
.text:00080...	00000010	unicode	3\t%S\t%S
.text:00080...	0000001E	unicode	Continue Trial
.text:00080...	0000001C	unicode	Trial Expired
.text:00080...	00000010	unicode	Buy Now
.text:00080...	00000016	unicode	Days Left
.text:00080...	00000010	unicode	0\t%S\t%S
.text:00080...	00000008	unicode	Buy
.text:00080...	00000018	unicode	Buy Via Wap

كما تلاحظ فإن كل ما يبحث عنه الكراكر موجود هنا، ببساطة وبلا تعقيد علينا أن نحصل على Licensed User بدلاً من Trial Expired أو Continue Trial ، لذلك قم بالضغط مرتين على Licensed User لتصل إلى هنا :

```
.text:00080592 00 00                         DCW 0
.text:00080594 49 00 6E 00+    aInvalidKeyPlea unicode 0, <Invalid Key! Please tr
.text:00080594 76 00 61 00+                                          ; DATA XRE
.text:000805D2 00 00                         DCW 0
.text:000805D4 4C 00 69 00+    aLicensedUser   unicode 0, <Licensed User>,0
.text:000805D4 63 00 65 00+                                          ; DATA XRE
.text:000805D4 6E 00 73 00+                                          ; .text:of
.text:000805F0 43 00 6F 00+    aContinue___    unicode 0, <Continue...>,0
.text:000805F0 6E 00 74 00+                                          ; DATA XRE
```

ضع مؤشر الفأرة على aLicensedUser ثم اضغط "X" واختر الخيار الأول لتصل إلى هنا :

```
.text:000300D4
.text:000300D4                           ; ------------------------------------
.text:000300D8 D4 05 08 00       off_300D8        DCD aLicensedUser
.text:000300D8
.text:000300DC F0 05 08 00       off_300DC        DCD aContinue___
.text:000300DC
.text:000300E0 08 06 08 00       off_300E0        DCD off_80608
.text:000300E0
.text:000300E4 18 06 08 00       off_300E4        DCD aContinueTrial
.text:000300E4
.text:000300E8 38 06 08 00       off_300E8        DCD aTrialExpired
.text:000300E8
```

أيضا مؤشر الفأرة على off_300D8 ثم "X" لتصل أخيراً إلى هنا :

```
.text:0002FB78    LDR     R3, [R11,#var_20]
.text:0002FB7C    CMP     R3, #1
.text:0002FB80    BNE     loc_2FC24       ; *** This is the magic branch !
.text:0002FB84    SUB     R0, R11, #0x94
.text:0002FB88    BL      TDeseroEv
.text:0002FB8C    SUB     R0, R11, #0x104
.text:0002FB90    BL      TDeseroEv
.text:0002FB94    SUB     R0, R11, #0x174
.text:0002FB98    BL      TDeseroEv
.text:0002FB9C    SUB     R0, R11, #0x17C
.text:0002FBA0    LDR     R1, =aLicensedUser ; *** will be intercomputernet
.text:0002FBA4    BL      TPtrC16CEPKt
```

كما ترى يوجد قفزة تتعدى aLicensedUser وهي BNE : Branch if Not Equal لتصل إلى aContinuetrial أو إلى aTrialExpired حسب الحالة.

مبدئياً سنقوم بتغيير القفزة إلى NOP حتى لا يتم تنفيذها، وبالتالي يذهب البرنامج إلى aLicensedUser ، لذلك انتقل إلى الخطوة الثالثة وقم بتعديل البايتات اللازمة كما في الجدول :

عنوان الأوفست Offset	البايتات المعدلة	البايتات الأصلية
27C1C	00 00 A0 E1	27 00 00 1A

اذهب إلى الخطوة الرابعة ومن بعد استبدال الملف المعدّل بالأصلي قم بتشغيل البرنامج، وستلاحظ أنه قد تم تسجيله :



حسناً قم بتقديم التاريخ شهراً واحداً للأمام وافتح البرنامج ماذا تلاحظ ؟

لقد ظهرت رسالة Trial is Expired ولكن يتم فتح البرنامج وهو يعمل بكافة خصائصه، إذن فهذه مجرد رسالة مزعجة أي Nag Screen ولكن مع ذلك لا مانع من إخفائها، فلنعد إلى برنامج IDA وفي نافذة Strings ابحث عن جملة Trial is Expired واتبع نفس الخطوات التي عملناها سابقاً لتصل أخيراً إلى هنا :

```
.text:0000F368 280 51 0F 4B E2     SUB    R0, R11, #0x144
.text:0000F36C 280 AC 13 9F E5     LDR    R1, =aTrialIsExpired
.text:0000F370 280 33 8D 00 EB     BL     TPtrC16CEPKt     ;     *** التعديل الأول
.text:0000F374 280 51 3F 4B E2     SUB    R3, R11, #0x144
.text:0000F378 280 9C 01 1B E5     LDR    R0, [R11,#var_19C]
.text:0000F37C 280 01 10 A0 E3     MOV    R1, #1
.text:0000F380 280 03 20 A0 E1     MOV    R2, R3
.text:0000F384 280 72 8E 00 EB     BL     CONE_11          ;     *** التعديل الثاني
.text:0000F388 280 9C 31 1B E5     LDR    R3, [R11,#var_19C]
```

وهذا كل ما سنحتاجه لإلغاء تلك الرسالة المزعجة :

البايتات الأصلية	البايتات المعدلة	عنوان الأوفست Offset
33 8D 00 EB	00 00 A0 E1	740C
72 8E 00 EB	00 00 A0 E1	7420

كما تلاحظ هناك تعديل أول وتعديل ثانٍ يتم بجعل التعليمتين NOP حتى لا يتم النداء على تلك النافذة المزعجة وهكذا نستطيع أن نقول: مبارك علينا النسخة الكاملة !

إلى هنا نصل إلى ختام الدرس على أمل لقاء قادم إن شاء الله، ودمتم في رعاية الله.

5. مرفقات الدرس

1) ملفات sis للبرامج التي تم كسرها في هذا الدرس.
2) ملف نصي فيه الـــ Opcodes لبعض تعليمات ARM .
3) برنامج Petran لضغط وفك ضغط الملفات التنفيذية.
4) متصفح الملفات للجوال X-plore S60 3rd edition v1.30 .

6. روابط مفيدة للمزيد من المعلومات حول معالجات ARM ونظام التشغيل Symbian

http://en.wikipedia.org/wiki/ARM_architecture

http://www.arm.com/

http://en.wikipedia.org/wiki/Symbian

http://www.forum.nokia.com/info/sw.nokia.com/id/2c835bc6-4106-4563-9eb07f215b27e83c/S60_3rd_Edition_Cpp_Developers_Library.html

http://developer.symbian.com

http://www.newlc.com

كسر تطبيقات Windows Mobile

WINCE Reversing

1. مقدمة

1.1 المتطلبات

- ARM7-TDMI-manual-pt2.
- arm_edit_Disassembler.
- DuckHunt.
- Fun2Link.
- Handmark Yathzee 1.05.
- WinCE.

1.2 تعليمات معالج ARM

التعليمات واللواحق التي في الجداول التالية هي للاستئناس لا الحصر.

- التعليمات :

المعنى	التعليمة
جمع مع الحمل	ADC
	ADD
And	AND
القفز	B
مقارنة	CMP
EOR	EOR
MOV	MOV
MUL	MUL

- معاني لواحق التعليمات :

المعنى	الأعلام	اللاحقة	Hex
يساوي	Z=1	EQ	0000
لا يساوي	Z=0	NE	0001
أكبر أو يساوي	N=V	GE	1010
أصغر تماماً	N!=V	LT	1011
أكبر تماماً	Z=0 && N=V	GT	1100
أصغر أو يساوي	Z=1 || N!=V	LE	1101
دائماً	غير مهم	AL	1110

يمكن أن تضاف اللواحق إلى نهاية التعليمات؛ مثلاً:

- تعليمة B تعني القفز، ولكن يمكن إضافة اللواحق عليها فتصبح مثلاً :

اقفز إذا كان يساوي	BEQ
اقفز إذا كان لا يساوي	BNE
اقفز إذا كان أصغر تماماً	BLT

سنقوم بعرض بعض التعليمات بصيغتها الثنائية :

تعليمة MOV :

Mov = XX XX A0 EX

رقم المسجل أو القيمة	XX
نوع عملية النقل X=1نقل قيمة مسجل إلى آخر X=3نقل قيمة مباشرة إلى مسجل	X

مثال :

02 03 A0 E1	MOV R3,R2

MOV R2,#5	**05 02 A0 E3**
MOV R0,R4	**04 00 A0 E1**
MOV R5,#FF	**FF 05 A0 E3**

تعليمة CMP :

CMP = XX 00 5X EX

رقم المسجل الثاني أو القيمة	XX
رقم المسجل الأول	X
نوع عملية المقارنة X=1مقارنة قيمة مسجل مع آخر X=3مقارنة قيمة مباشرة مع مسجل	X

مثال :

CMP R2,R3	**03 00 52 E1**
CMP R4,#1	**01 00 54 E3**
CMP R0,R4	**04 00 50 E1**
CMP R5,# FF	**FF 00 55 E3**

تعليمة BRANCH (B) :

BRANCH = XX XX XX EA
BEQ: If Z = 1 (XX XX XX 0A)
BNE: If Z = 0 (XX XX XX 1A)
BMI: If N = 1 (XX XX XX 4A)

شيء أخير :

CMP R0, R1

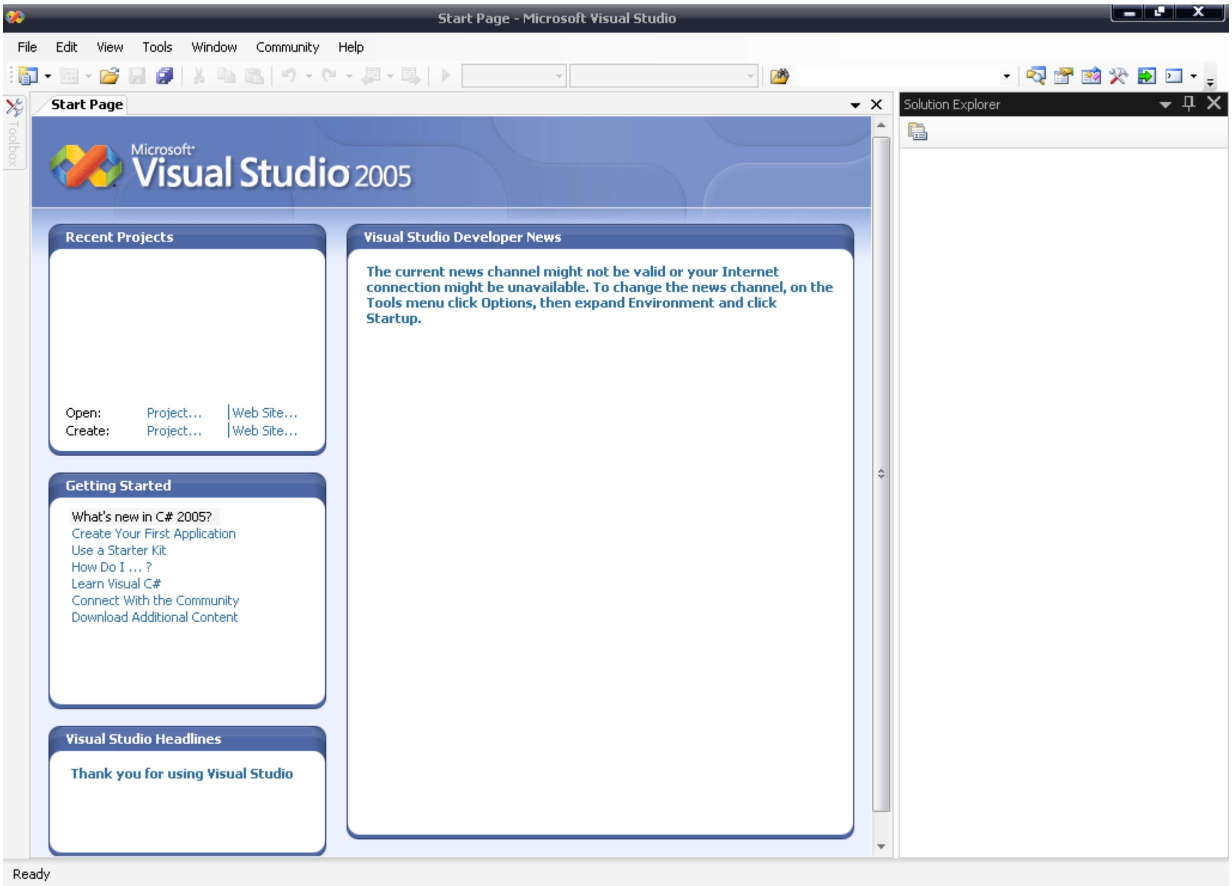
N=1	Z=1	C=1
R0 >= R1	R0=R1	R0>=R1

للمزيد من المعلومات : انظر ARM7- DATA SHEET في مرفقات الدرس

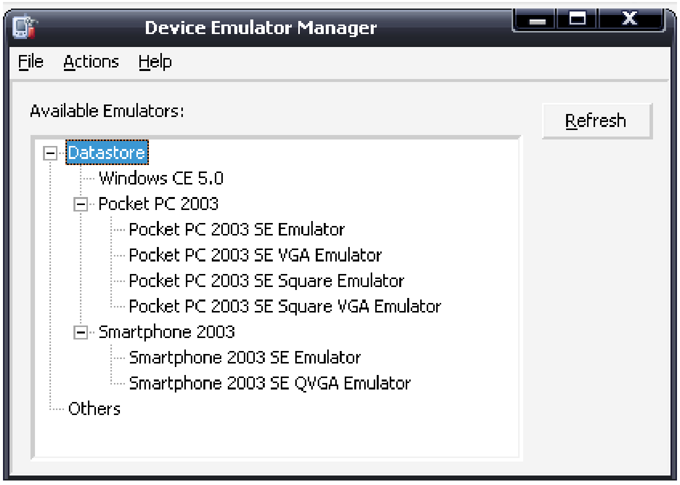
2. محاكاة عمل تطبيقات الـ Pocket PC على الكمبيوتر

- Microsoft Visual Studio 2005.
- Microsoft ActiveSync 4.5.

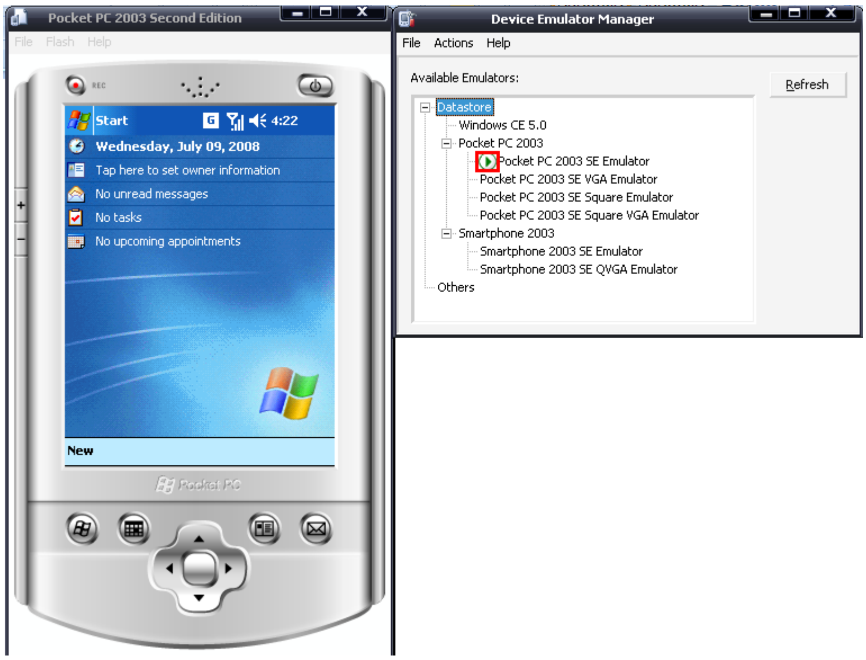
أولاً شغل : Microsoft Visual Studio 2005 لتظهر الشاشة التالية :



ثم بعد ذلك Tools - > Device Emulator manager :



يمكنك من هنا اختيار أي نظام لتحاكيه، ولكننا سنختار Pocket Pc 2003 ، الآن اضغط بالزر الأيمن للفأرة على SE Emulator Pocket Pc 2003 واختر Connect :



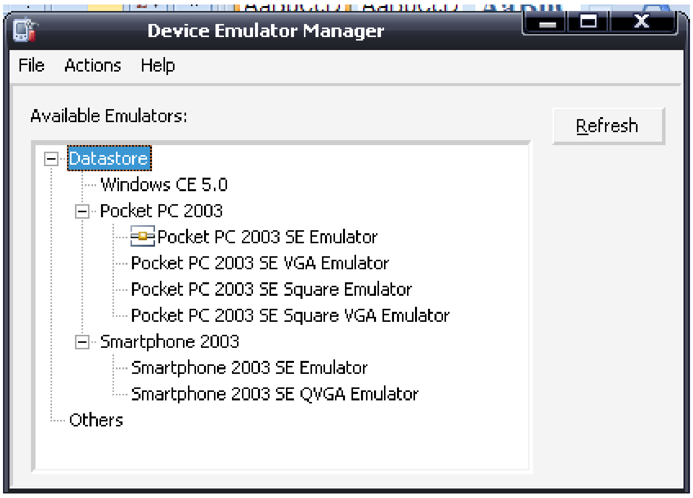
انتظر قليلاً.. وسيظهر كما في الصورة.



انتبه للإشارة داخل المربع الأحمر في الصورة!! فهي دليل الاتصال بالمحاكي.

الآن سنقوم بربط الجهاز مع Microsoft ActiveSync 4.5 وبهذا نستطيع تحميل البرامج على الجهاز المحاكي.

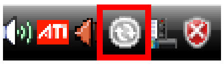
اضغط بزر الفأرة الثاني على Pocket Pc 2003 SE Emulator ثم اختر Cradle :

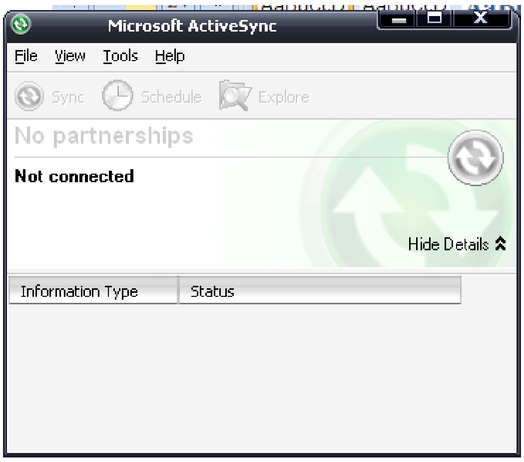




انتبه للإشارة أمام Pocket Pc 2003 SE Emulator.

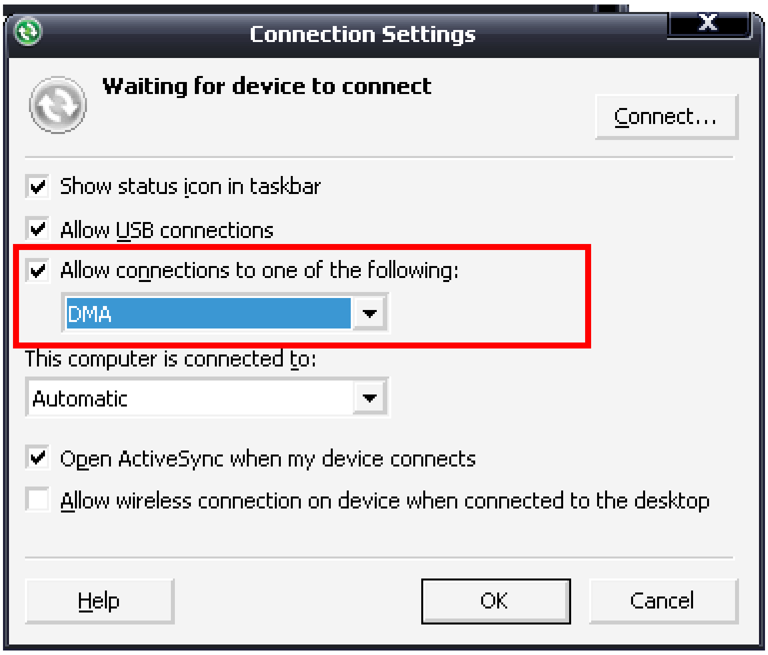
اضغط ضغطتين على Microsoft ActiveSync 4.5 الموجود في شريط المهام :



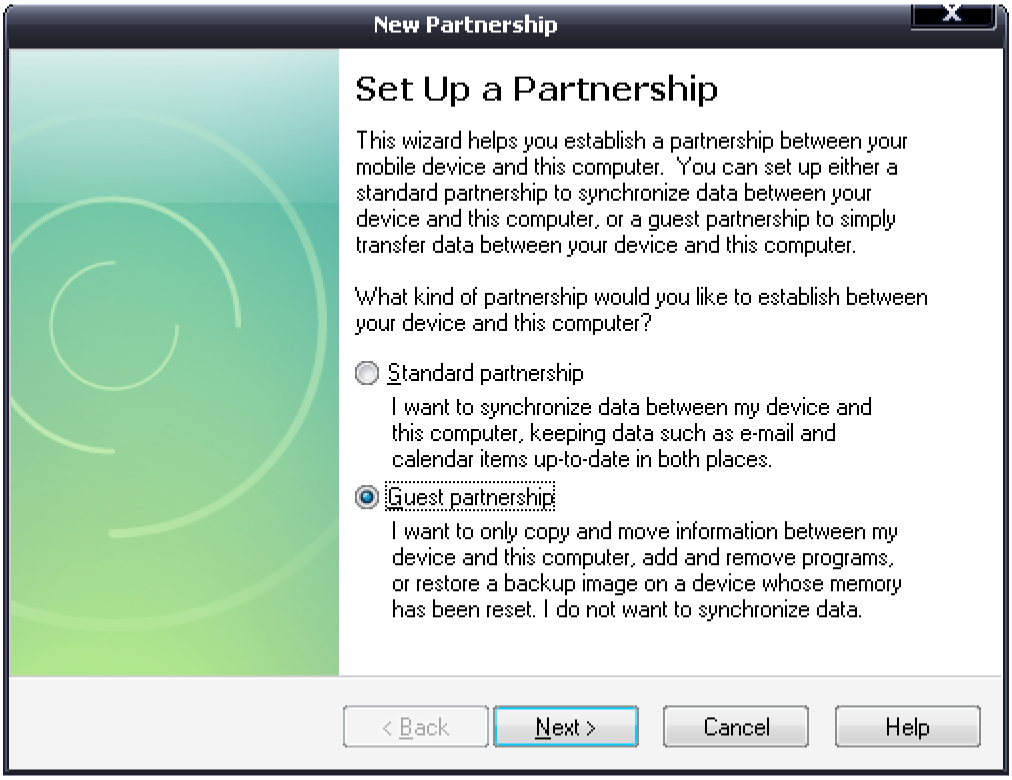


File -> Connection Settings

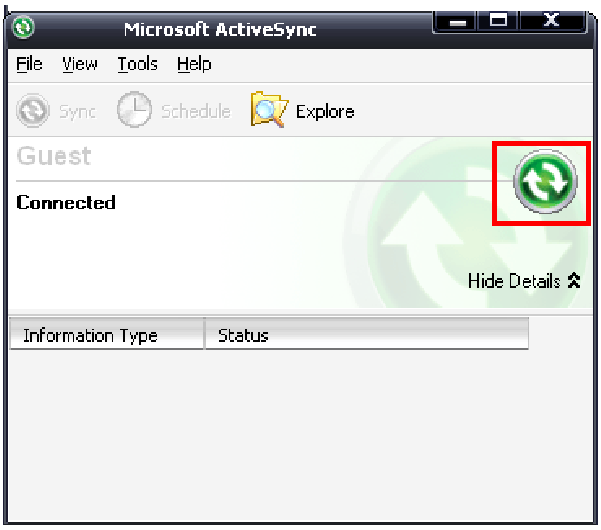
واضبط الإعدادات كما في الصورة :



انتظر قليلاً.. وسيتم الاتصال :



اختر Guest :





لاحظ الفرق بين هذه الصورة والصورة قبل تأسيس الاتصال.

الآن يمكننا تحميل أي برنامج على Emulator وحتى استعراض الملفات الموجودة فيه.

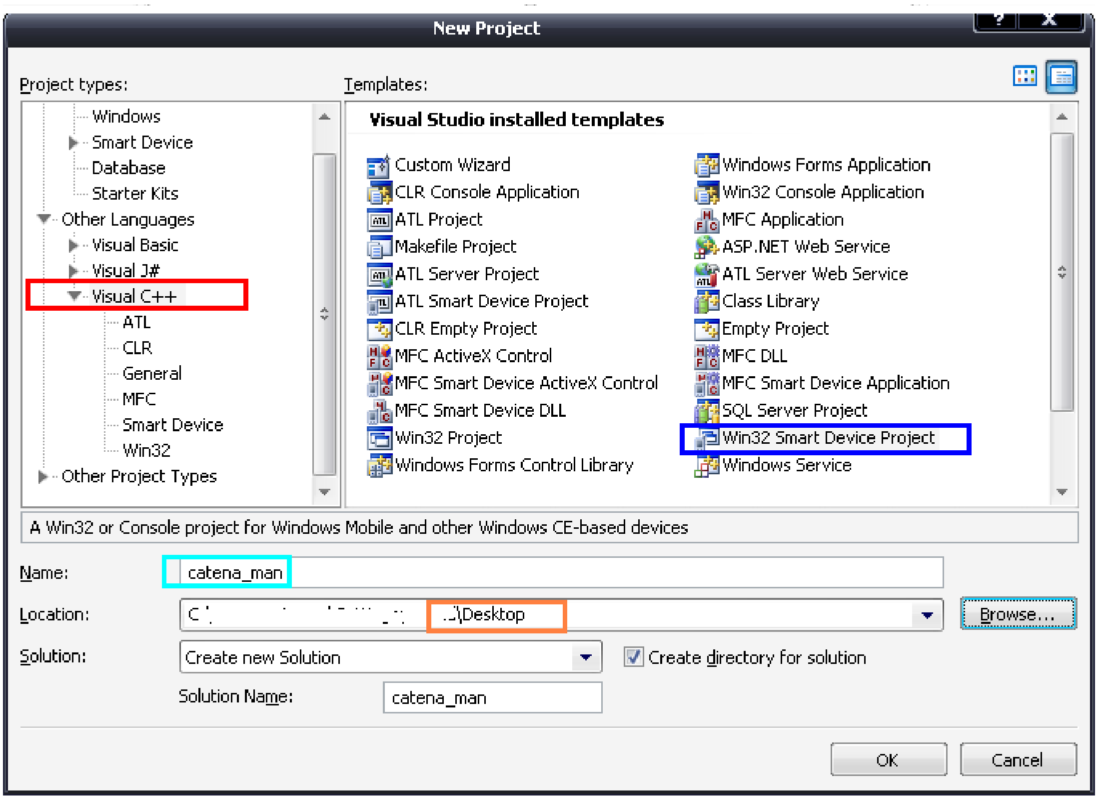
اضغط على Explore الموجودة في نافذة Microsoft ActiveSync .

ويمكننا حتى أن نقوم بعمليات الهندسة العكسية وحتى الـ Remote Debugger .

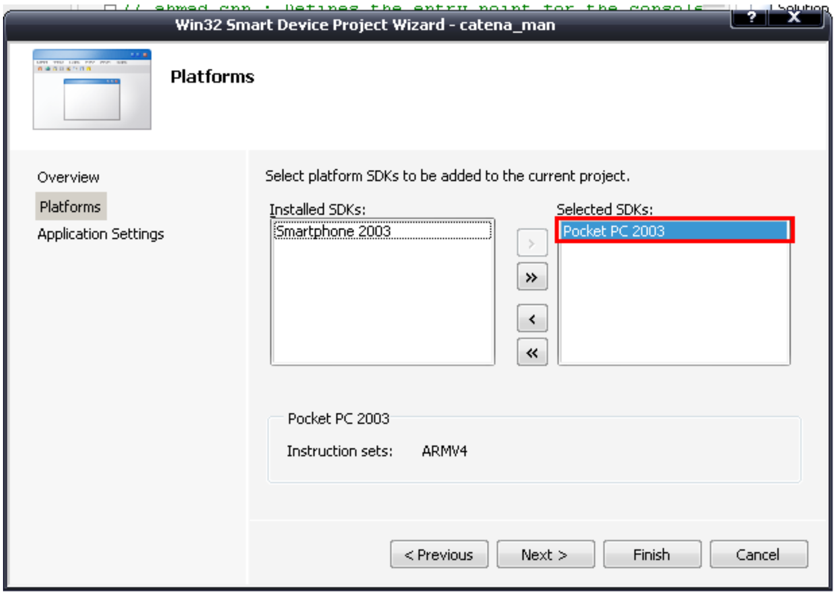
قم بجولة في نظام المحاكي وتمتع بإمكانيات Windows Mobile .

3. عمل Debugger باستخدام Microsoft Visual C++ 2005 وEmulator

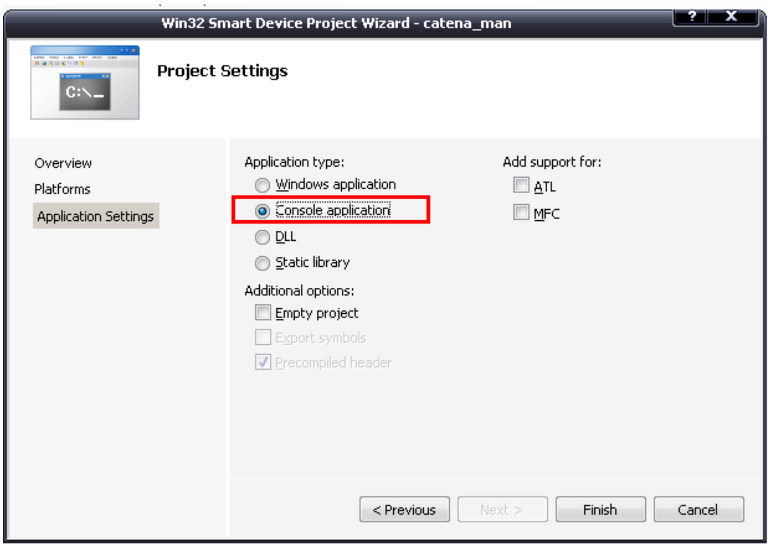
شغل Microsoft Visual Studio 2005 ثم New -> project واختر كما في الصورة :



Ok -> Next ->



Next ->

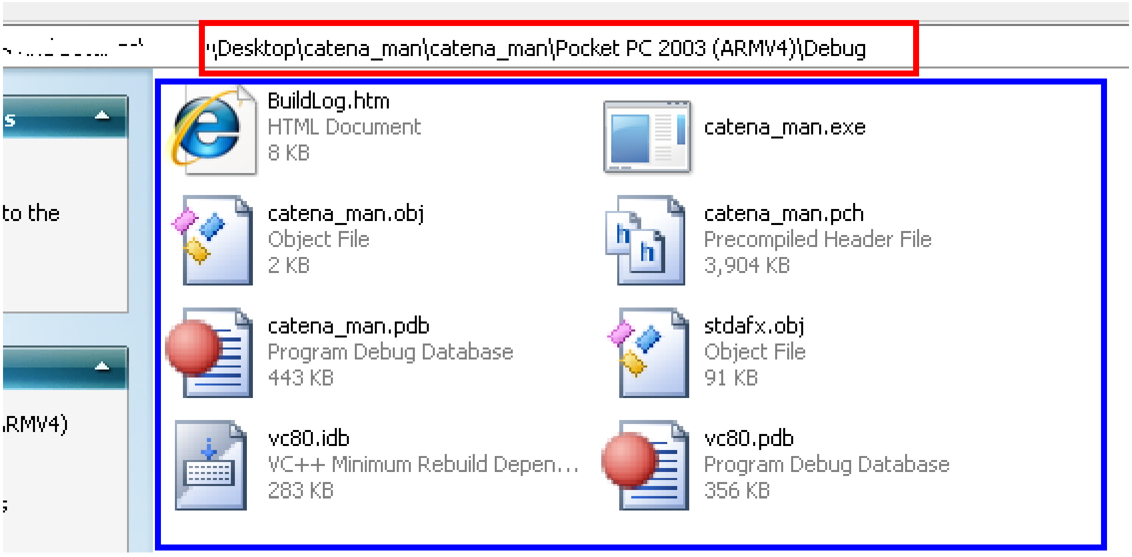


Finish.

الآن المشروع مشروع Console Smart Device فارغ ثم Build -> Build Solution



تأكد من المسار التالي ومن أن الملفات موجودة!!

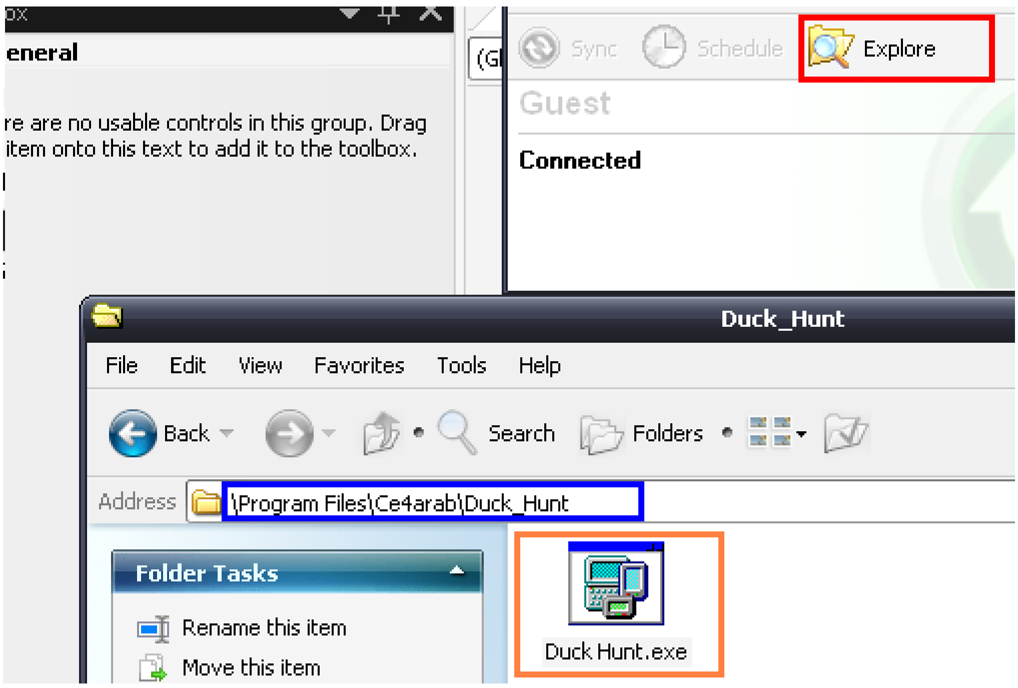


الآن سنقوم بعمل Debugger للعبة DuckHunt .

	اللعبة ليست بحاجة للكسر، فقط أضف اسمك لـ Owner وستعمل، ولكن للحاجة في أنفسنا قمنا بذلك.

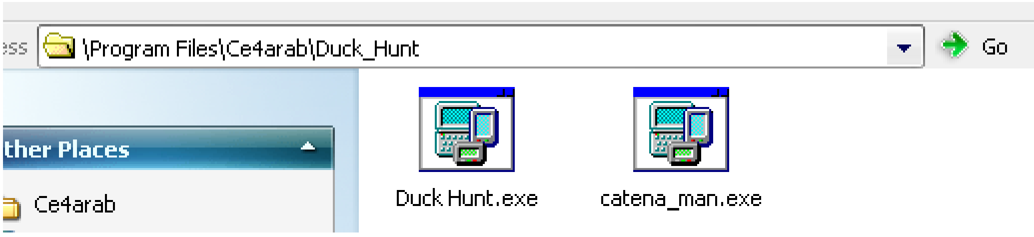
قم بالمحاكاة كما فعلنا في الفقرة السابقة وحمل اللعبة على الـــ Emulator .

سنقوم بعمل Remote Debugger على اللعبة، ولكن قبل ذلك لننسخ مسار اللعبة شغلMicrosoft ActiveSync 4.5 ثم اضغط على Explore وتابع فتح الملفات حتى تصل إلى المسار الموضح بالصورة :

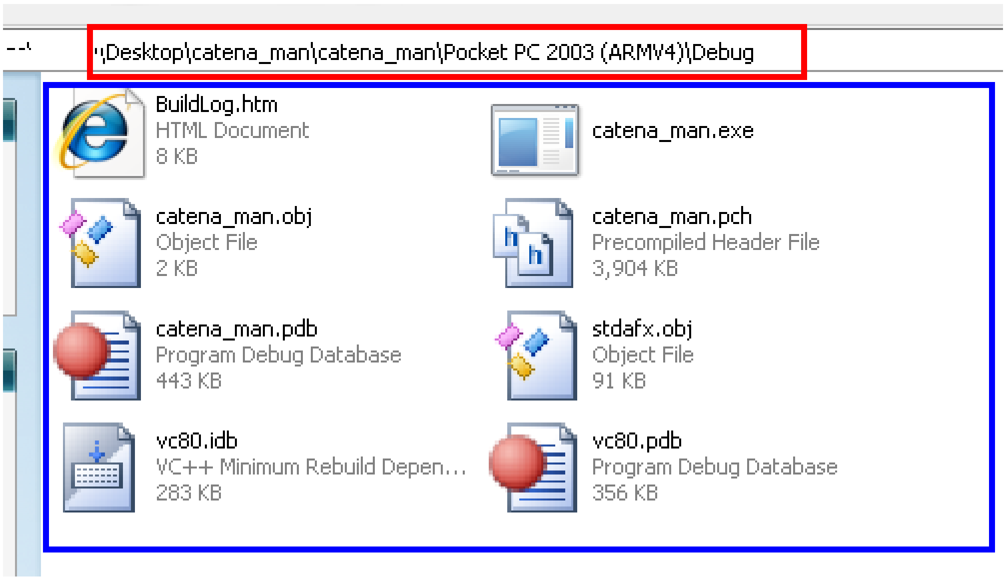


انسخ اللعبة وغير اسمها إلى catena_man .

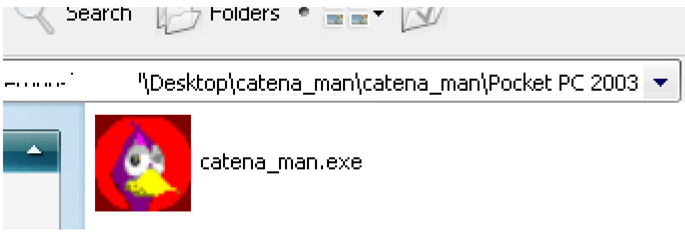
catena_man كما في اسم المشروع .



أتذكر هذه الصورة؟

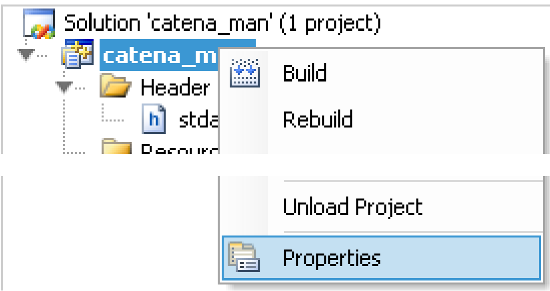


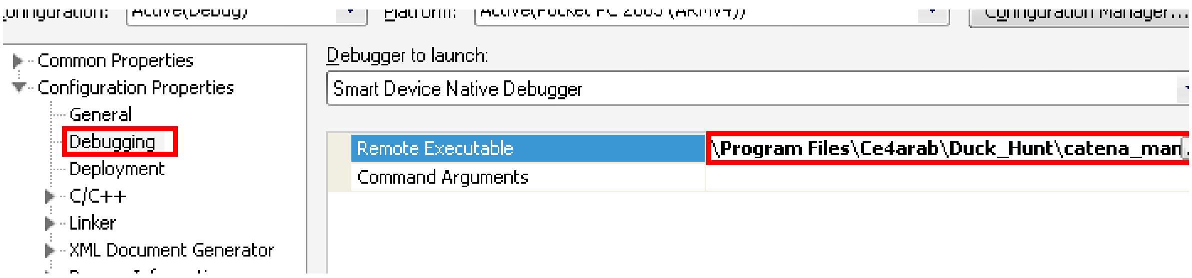
ستصبح :



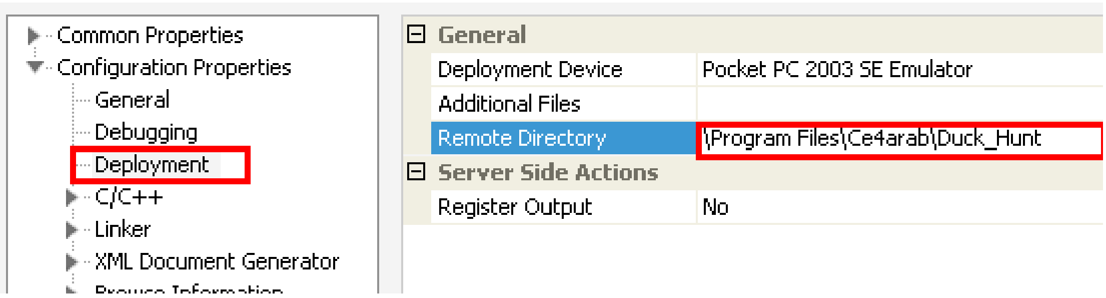
- ما قمنا به هو حذف كل الملفات داخل مجلد Debug ونسخ الملف catena_man إلى المجلد.
- لا ترى أن تحفظ مسار اللعبة على Emulator.

عد إلى Microsoft Visual Studio 2005 :



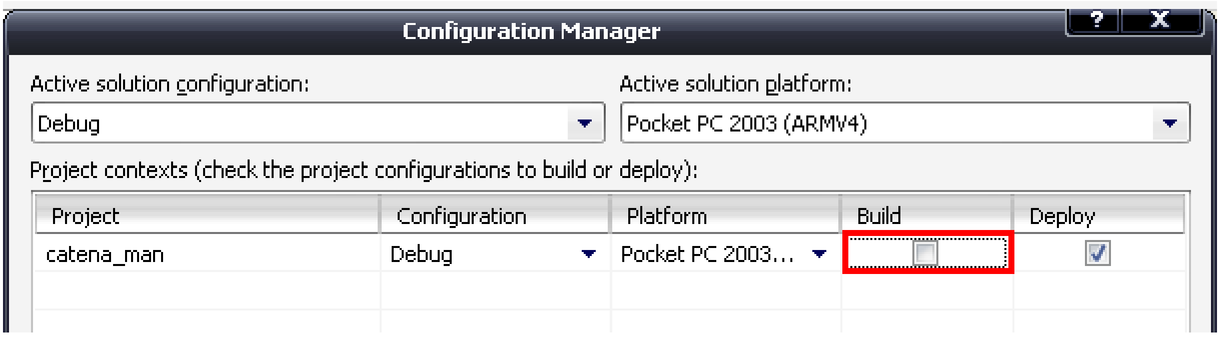


المسار: \ Program Files\Ce4arab\Duck_Hunt\catena_man.exe

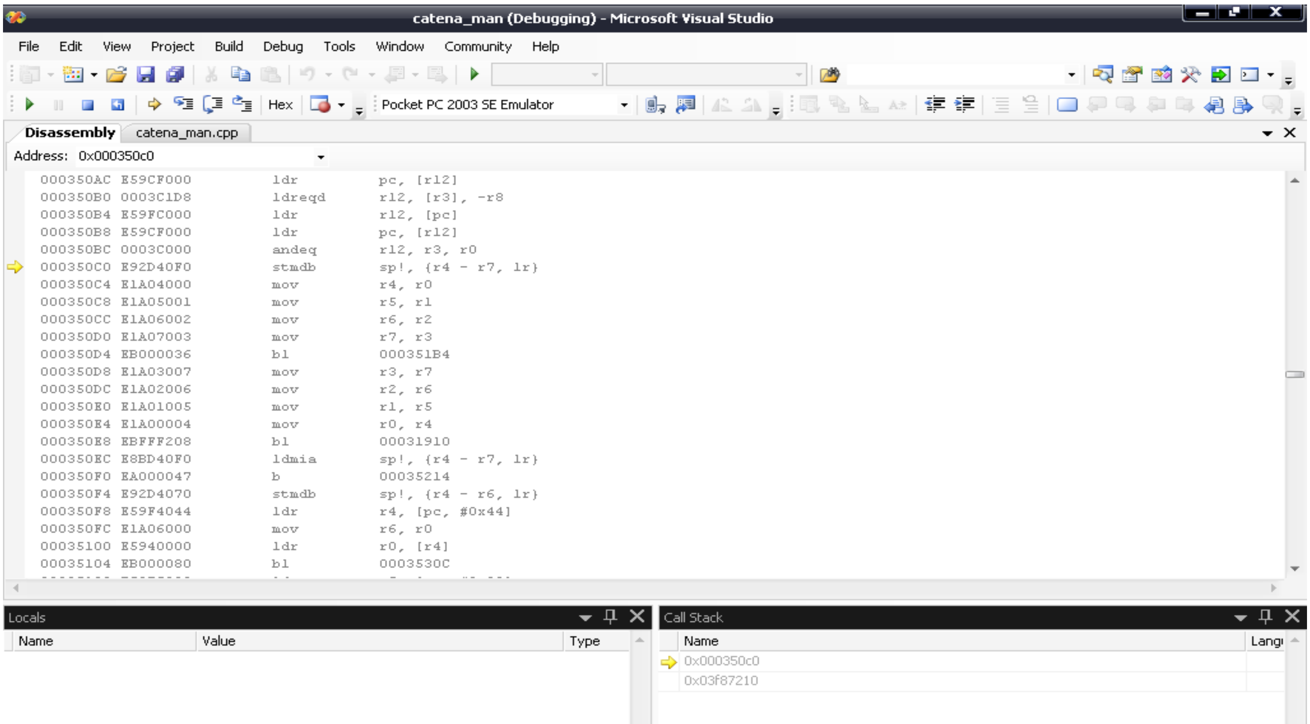


لا تكتب اسم البرنامج عند التعامل مع Deployment .

و الآن، اضغط على Configuration Manager :



Close - > Ok والآن F11 وسترى :



حاول أن تستمر بالتتبع ولاحظ أن الــ Emulator يعمل مع أوامر الــ Debugger

لماذا تم اختيار هذه اللعبة؟

لاحظ أن هذا اللعبة تتكون من ملف واحد فقط، وهو ملف الــ exe ففي هذه الحالة لم نكن بحاجة إلى ضبط إعدادات الــ Remote Debugger بل كان ممكن الاكتفاء بنسخ اللعبة إلى مجلد الــ Debug وتشغيل البرنامج (F11) وحتى دون تشغيل الــ Emulator وMicrosoft Visual Studio2005 سيتولى تشغيل الــ Emulator ونسخ اللعبة.

- هذه الطريقة لا تنفع مع البرامج التي تحتوي مكاتب ربط أو ملفات داتا أو صور أو غيرها.
- يمكن الاستغناء عن الــ Emulator وذلك باستخدام جهاز Pocket pc حقيقي واتباع الخطوات السابقة و سترى أن الــ Debugger سينفذ على جهازك الحقيقي .

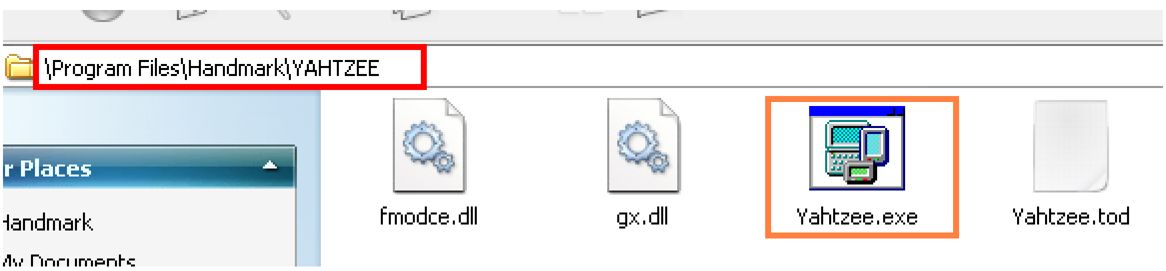
4. عمل Debbuge باستخدام IDA و Emulator

- IDA.
- WinCE Plugin For IDA (مع مرفقات الدرس):
  ← انسخ الملفات ida_kdstub.dll وwince_remote_arm.dll إلى المجلد الأساسي لــ IDA
  وانسخ الملف wince_stub.plw الموجود داخل ملف الــ Plugins إلى داخل ملف Plugins الموجود في مجلد IDA.
- Handmark Yathzee 1.05 (في المرفقات).

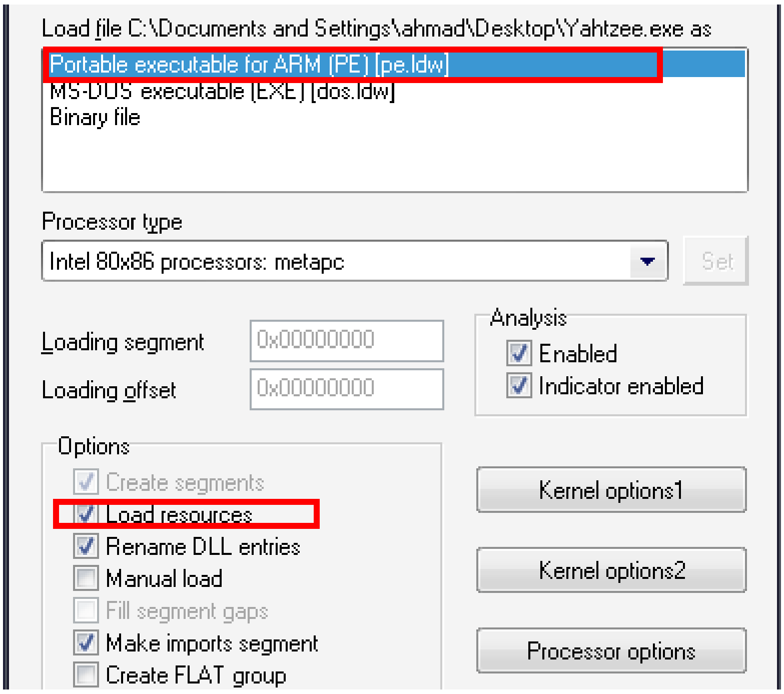
الآن سنقوم بعمل Debugger للعبة Handmark Yathzee 1.05 :

قم بالمحاكاة كما فعلنا في الفقرة السابقة وحمل اللعبة على الــ Emulator .

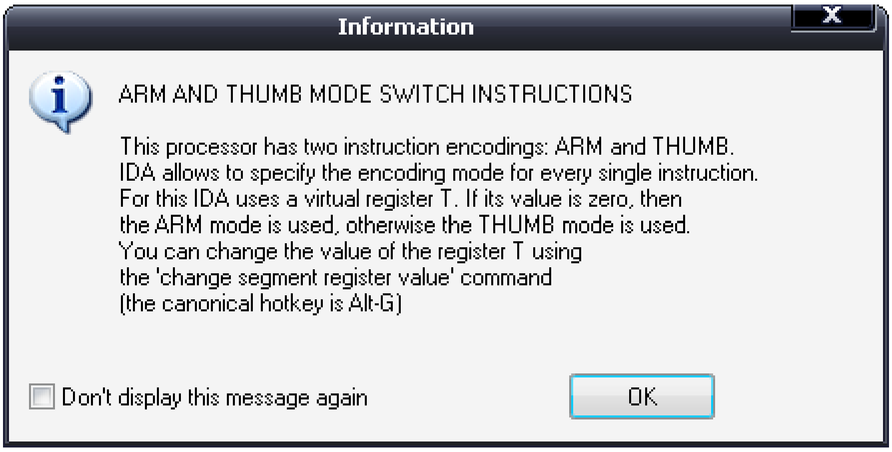
سنذهب ونحضر ملف اللعبة الــ EXE من Emulator ثم نشغل Microsoft ActiveSync 4.5 ثم نضغط على Explore ونتابع فتح الملفات حتى نصل إلى المسار الموضح بالصورة :



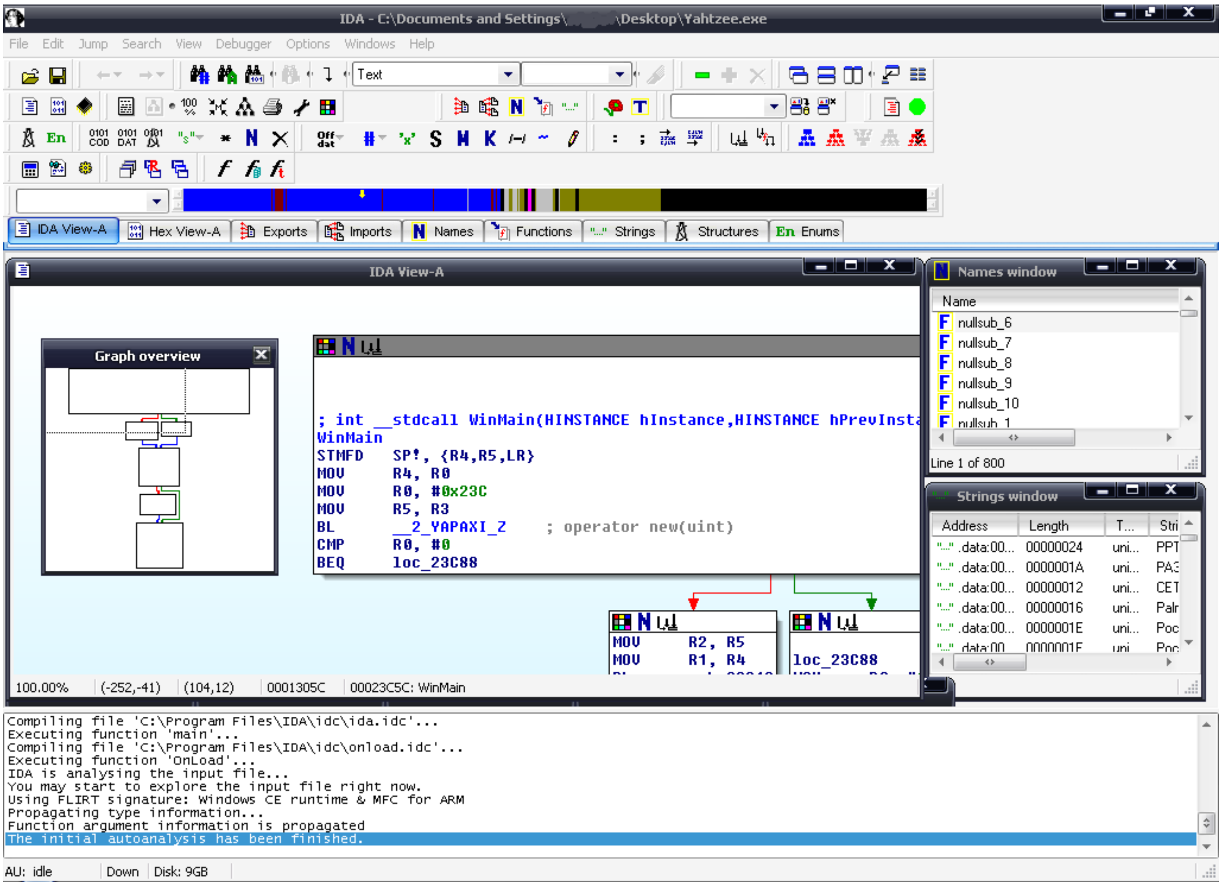
انسخ ملف اللعبة إلى سطح المكتب، ثم شغل IDA ثم اسحب اللعبة لداخله ستظهر لك النافذ اضبط الإعدادات :



اضغط ok :



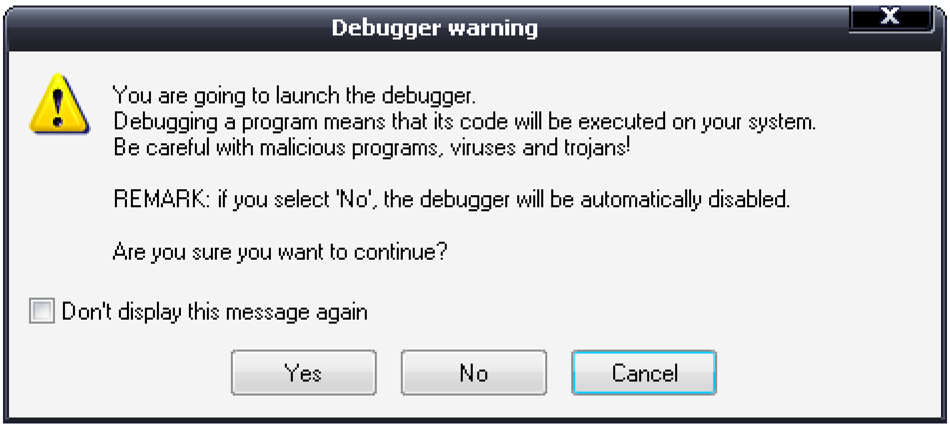
اضغط ok :

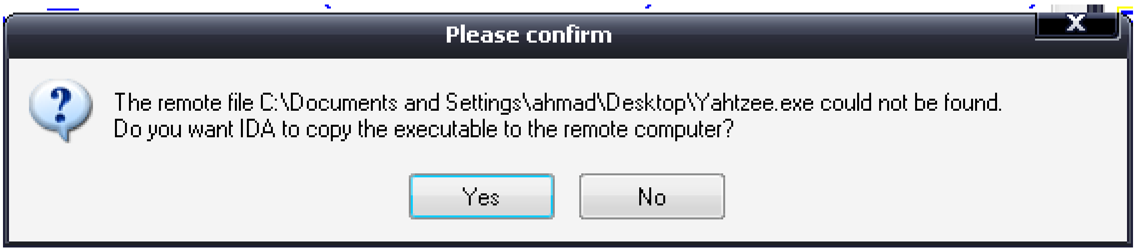


رائع انتهينا !!

Debugger -> Start process ثم OK

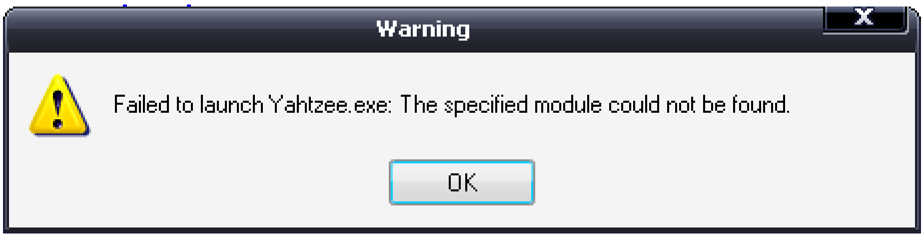
يجب أن يكون الـ Emulator في وضع العمل .





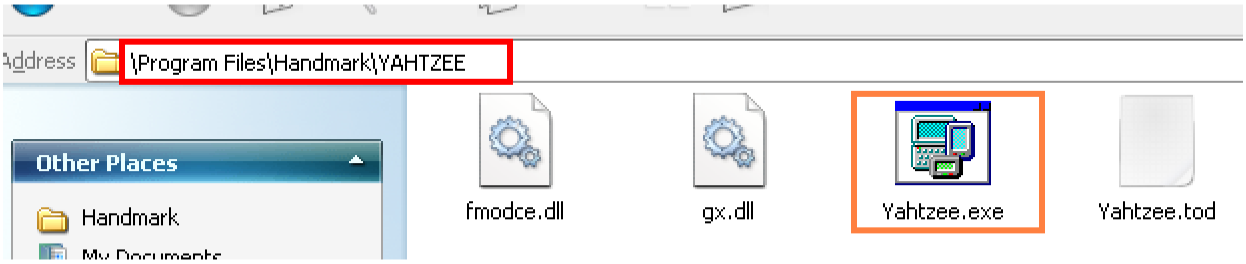
ما سنقوم به سيترتب عليه خطـــأ ولكن دعنا نتعلم!!

اضغط yes وعندها سينقل IDA ملف اللعبة الـــ EXE إلى المجلد الجذر في الـــ Emulator :



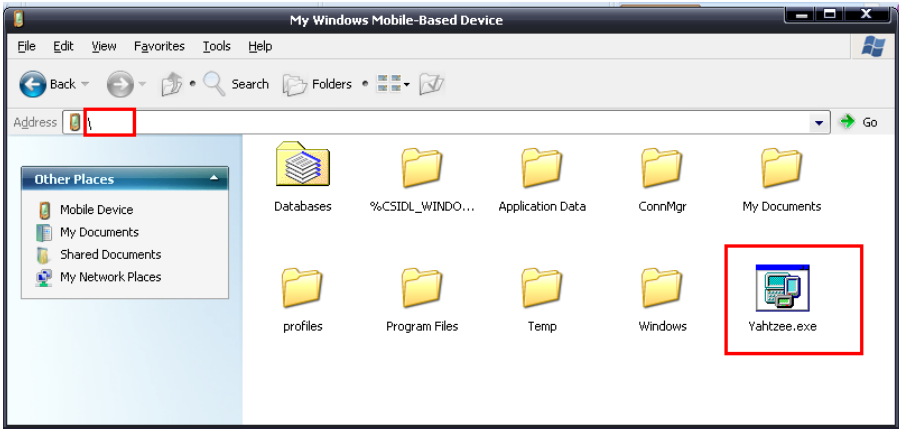
ما المشكلة ؟

أتذكر عندما تم نسخ اللعبة من ملفها انظر :



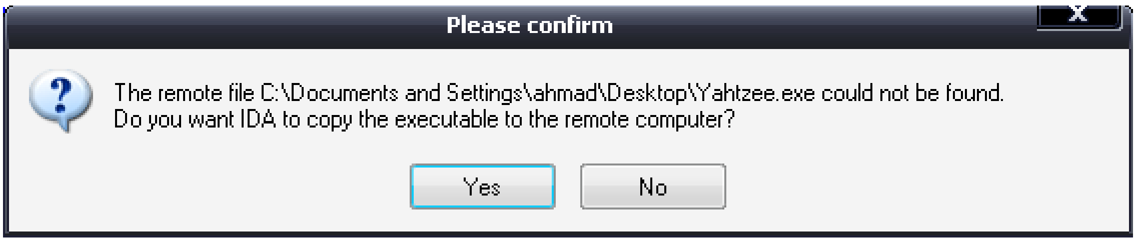
كان معها ملفات ربط تحتاجها كي تعمل، وعندما نسختها إلى سطح المكتب نسختها لوحدها.

لن تحل المشكلة بنسخ كل الملفات إلى سطح المكتب؛ لأن IDA لن ينسخ ملف الـــ EXE إلى الجذر الرئيسي :

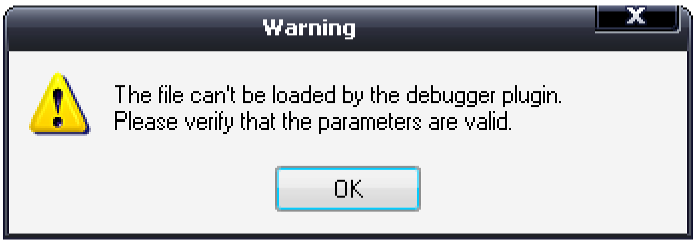


- قد تعمل اللعبة ولا تظهر الرسومات والأصوات لنفس السبب السابق .
- لو كانت اللعبة مكونة من ملف واحد فقط لنجحت العملية كما في اللعبة السابقة .

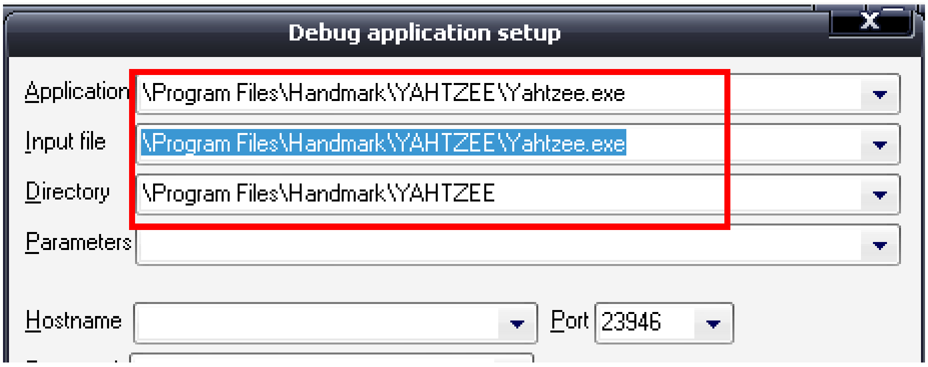
الآن أعد الكرة، Debugger -> Start process ولا تنس حذف الملف من الجذر الرئيسي المبين بالأعلى :



الآن NO

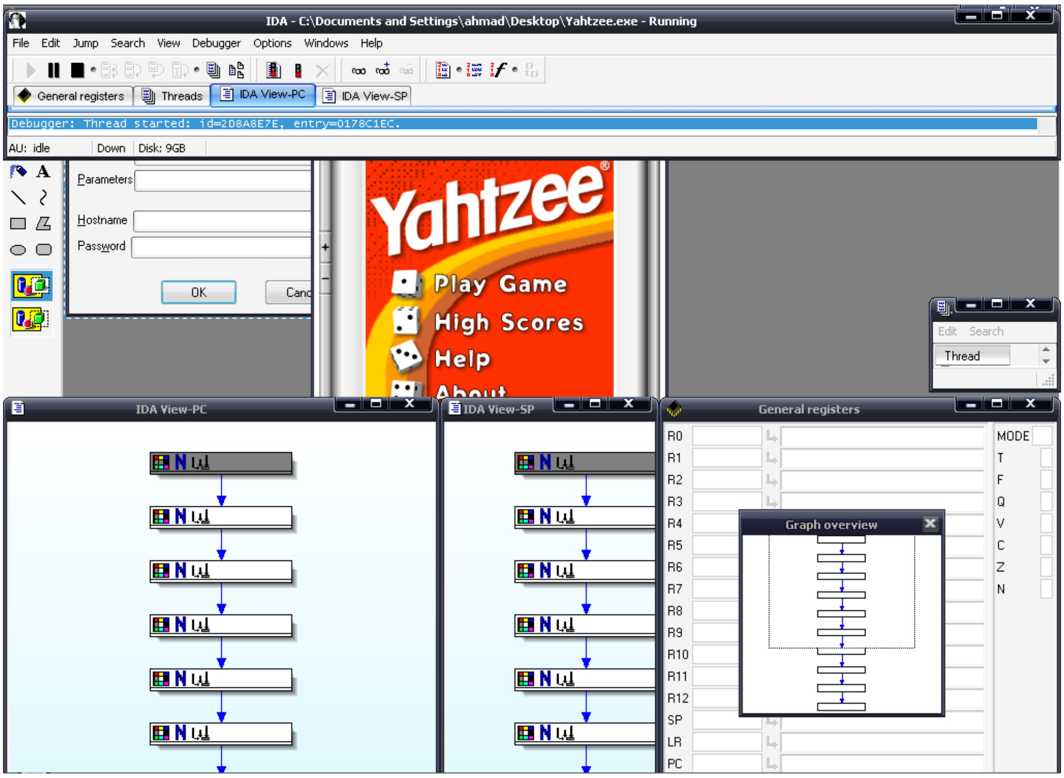


OK



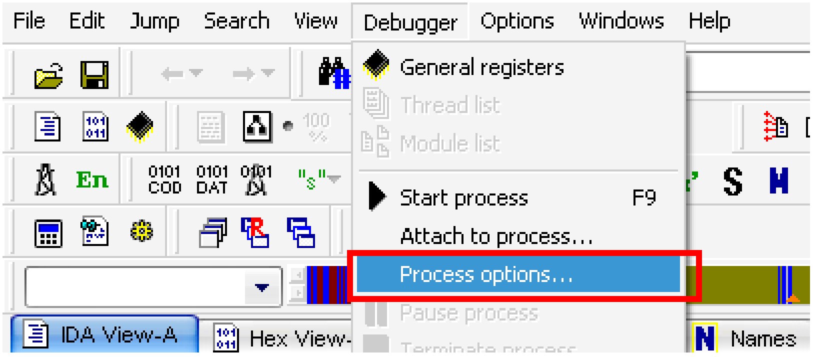
نفس المسارات في Emulator :

OK



رائع نجحنا والحمد لله!!

من الممكن ضبط إعدادات الـــ Remote Debugger قبل التشغيل، لاحظ :



كان من الممكن نسخ ملفات اللعبة إلى المكان الذي ينسخ فيه IDA الملف، جرب نسخ الملفات وستعمل اللعبة .

5. امثلة تطبيقية

5.1 المثال الأول : كسر لعبة Handmark Yathzee 1.05

اللعبة هي التي تم شرح الفقرة السابقة عليها، فلنجرب اللعبة على الـــ Emulator شغل اللعبة من :

Start -> Program -> games -> Yathzee

جرب أن تلعب، ستظهر رسالة تطلب التسجيل، أدخل أي اسم ستظهر الرسالة :

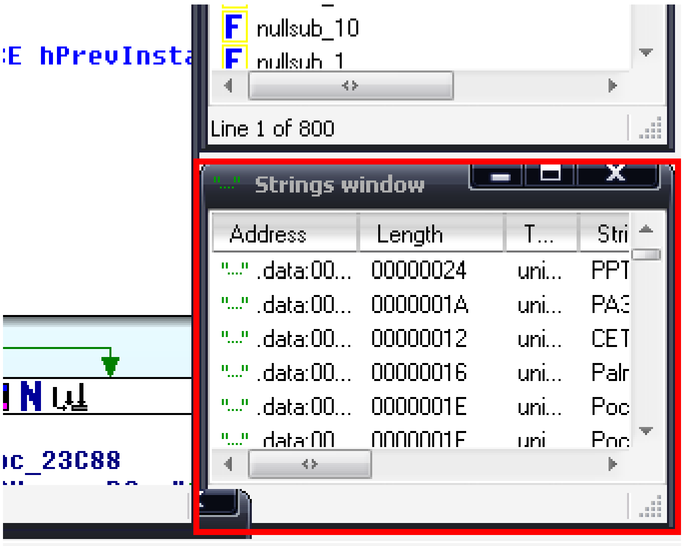


احفظ نص الرسالة فسيلزمك.

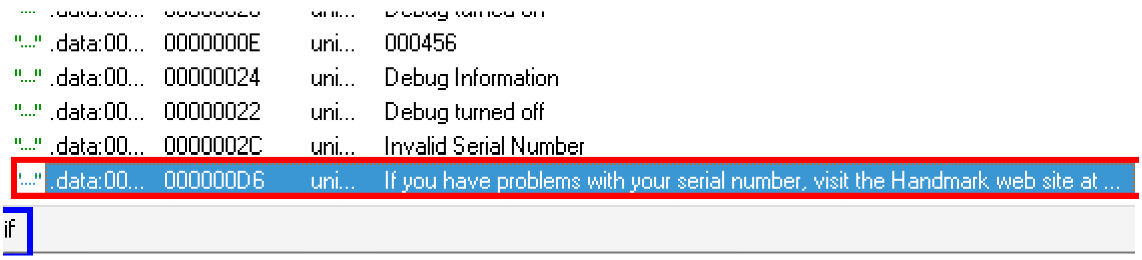
حمل اللعبة لـــ IDA ولا تقوم بعمل Debugger.

لا تنس يجب تشغيل الـــ Emulator وربطه مع Microsoft ActiveSync 4.5 وتحميل اللعبة.	

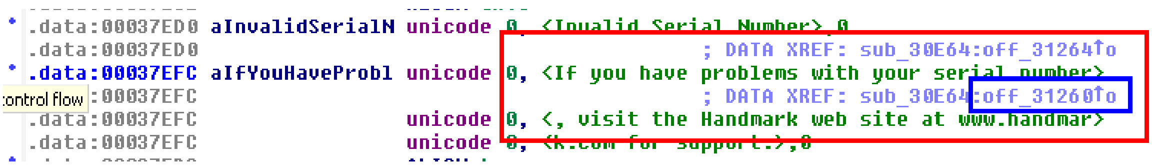
كبر نافذة Strings Windows :



ثم ابحث عن نص الرسالة، ويمكن أن تبدأ بالكتابة وIDA سيبحث (اكتب if) :



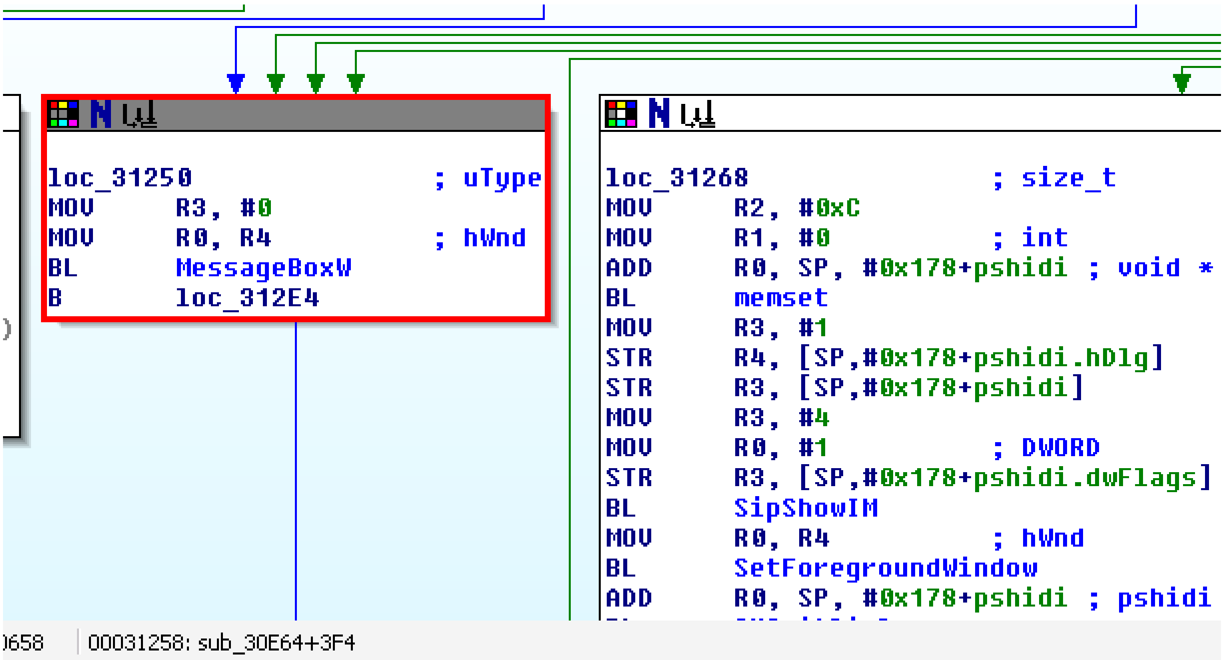
رائع لقد وجدنا نص الرسالة. اضغط ضغطتين على النص :



الآن اضغط ضغطتين عند المربع الأزرق لنذهب إلى مكان الكود الذي استخدمت فيه الجملة :

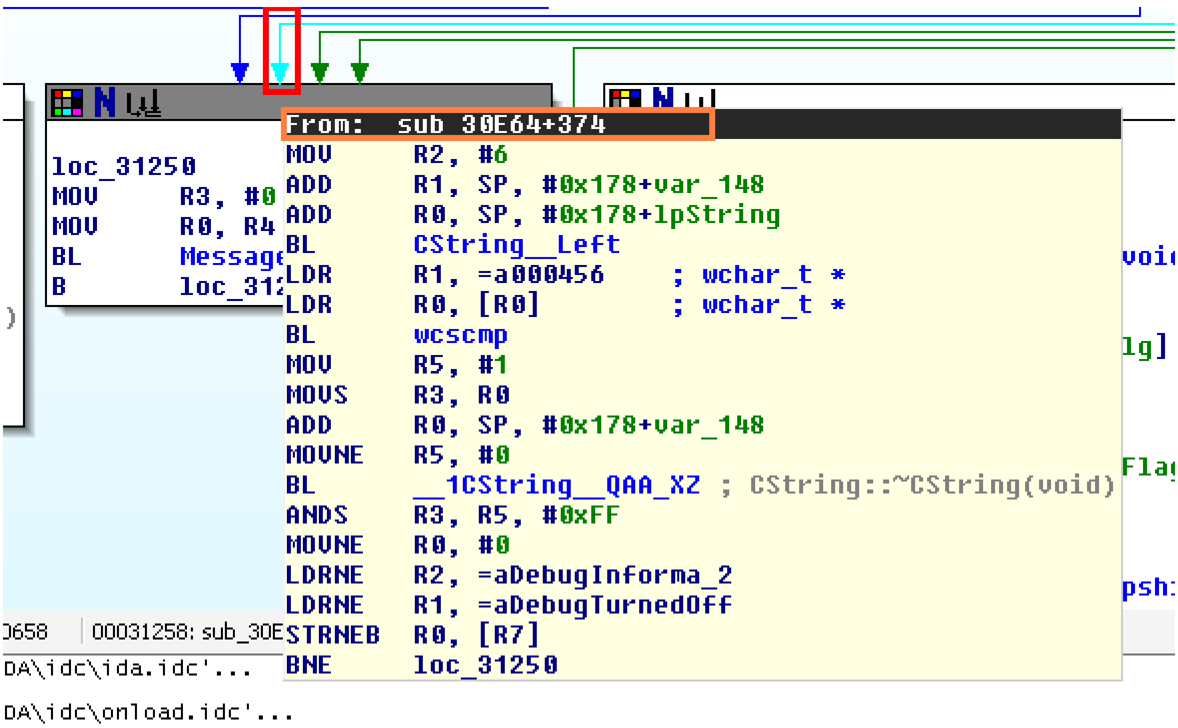
```
0003125C ;
00031260 ; LPCWSTR off_31260
00031260 off_31260       DCD aIfYouHaveProbl     ; DATA XREF: sub_30E64+3E8
00031260                                         ; "If you have problems wi
00031264 ; LPCWSTR off_31264
00031264 off_31264       DCD aInvalidSerialN     ; DATA XREF: sub_30E64:loc
00031264                                         ; "Invalid Serial Number"
00031268 ;
00031268
00031268 loc_31268                               ; CODE XREF: sub_30E64+38↑
00031268                 MOV     R2, #0xC        ; size_t
0003126C                 MOV     R1, #0          ; int
00031270                 ADD     R0, SP, #0x178+pshidi ; void *
00031274                 BL      memset
00031278                 MOV     R3, #1
0003127C                 STR     R4, [SP,#0x178+pshidi.hDlg]
```

حسناً هذا هو الكود، اضغط بزر الفأرة (في مكان لا يحوي تعليمات) الثاني واختر Graph Veiw :

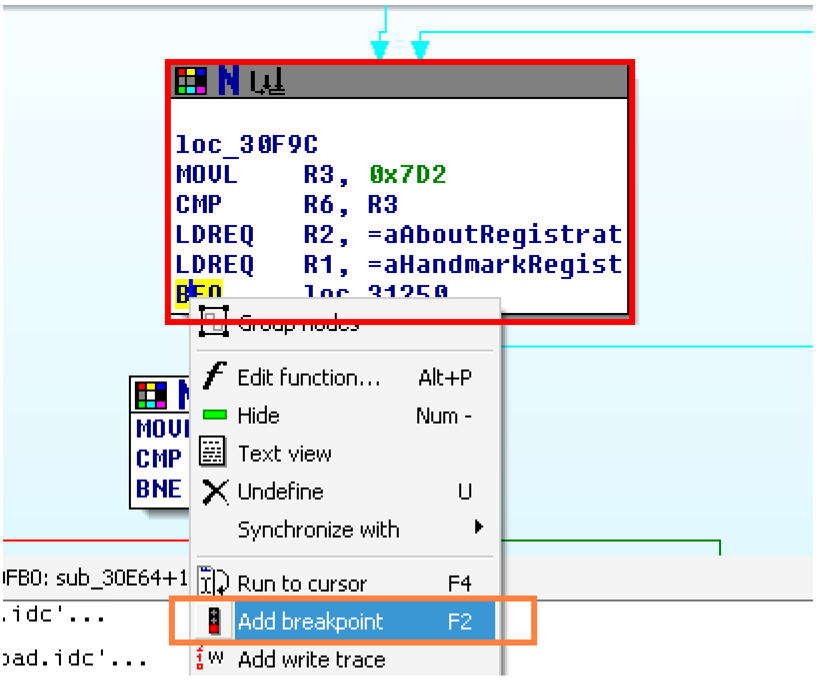


داخل المربع الأحمر الرسالة الغير مرغوبة، ويجب أن لا نصل إليها، ولا حظ أنه يتم الوصول إليها من أربع طرق، لاحظ الأسهم فوق المستطيل.

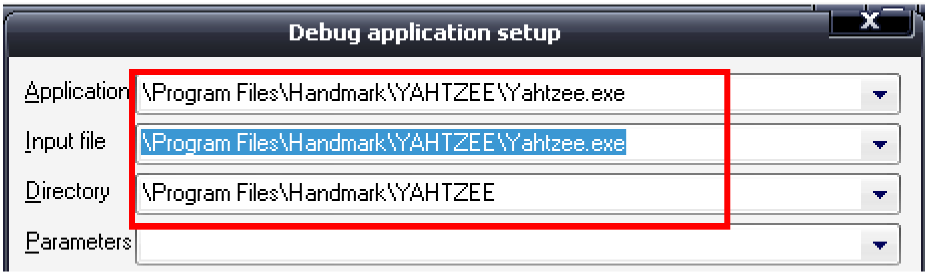
حسناً سنضع نقطة توقف على أبعد جزء من الكود الذي يوصلنا إلى الرسالة (أول جزء بالتحقق)، وسنعرفه من خلال الوقوف على الأسهم ومشاهدة العناوين أعلى المربعات :



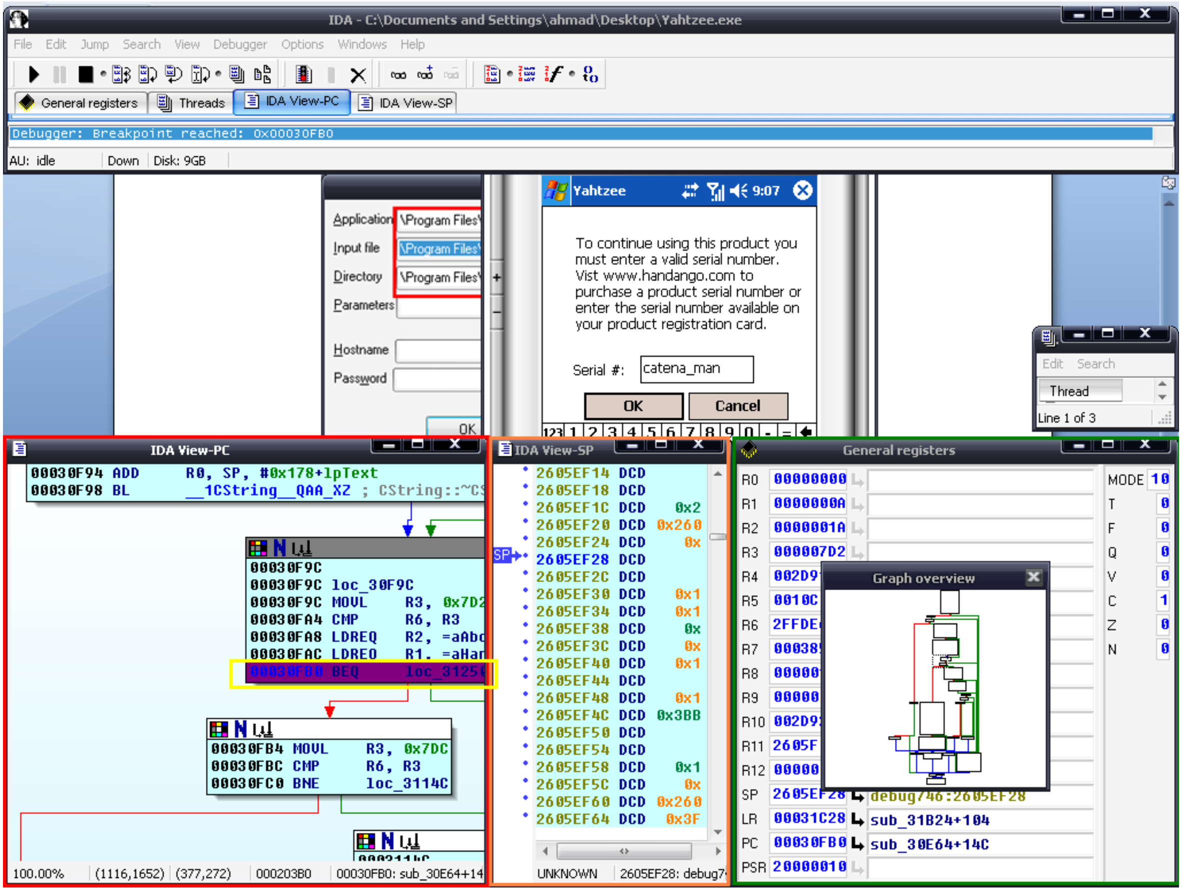
تفحص العناوين للأسهم كما في المربع البرتقالي سابقاً، واختر أقل هذه العناوين. طبعا ستجد أن أقل العناوين هو الأول من على اليمين اضغط على السهم مرتين :



ضع نقطة توقف ستصبح التعلٌمة باللون الأحمر، وإبدأ التتبع Debugger -> Start Process :



اكتب أي رقم تسجيل وانظر :

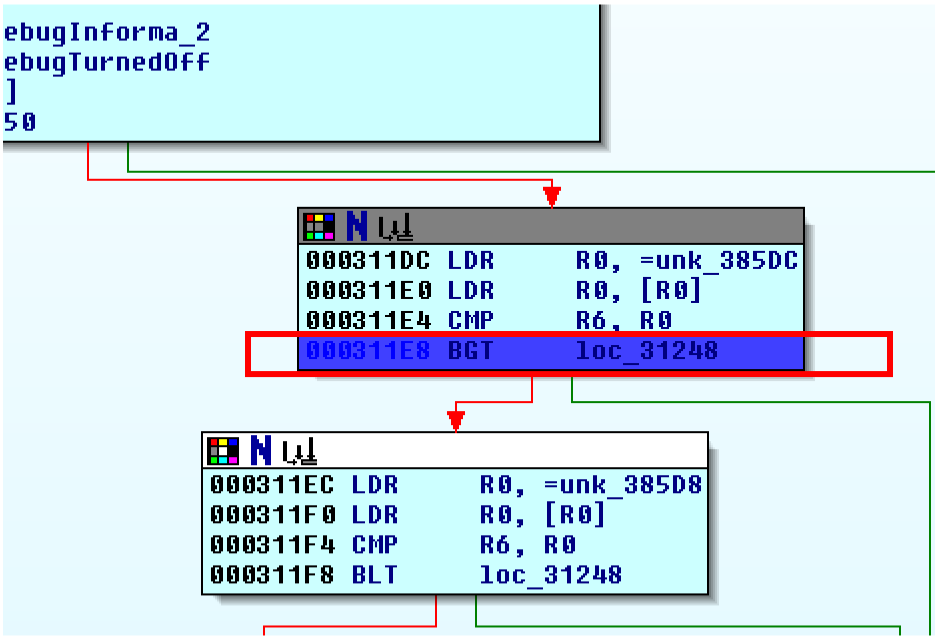


- الأحمر : هو نافذة التعليمات، ولا حظ أن IDA أوقف البرنامج عند نقطة التوقف.
- البرتقالي : نافذة الستاك.
- الأخضر : نافذة المسجلات.

للاستمرار بالتتبع سنضغط على F8 ولاحظ أن هذه القفزة لن تشكل مشكلة أثناء عملية التسجيل؛ لأنها لن تقودنا للرسالة (لاحظ ذلك من خلال وميض السهم).

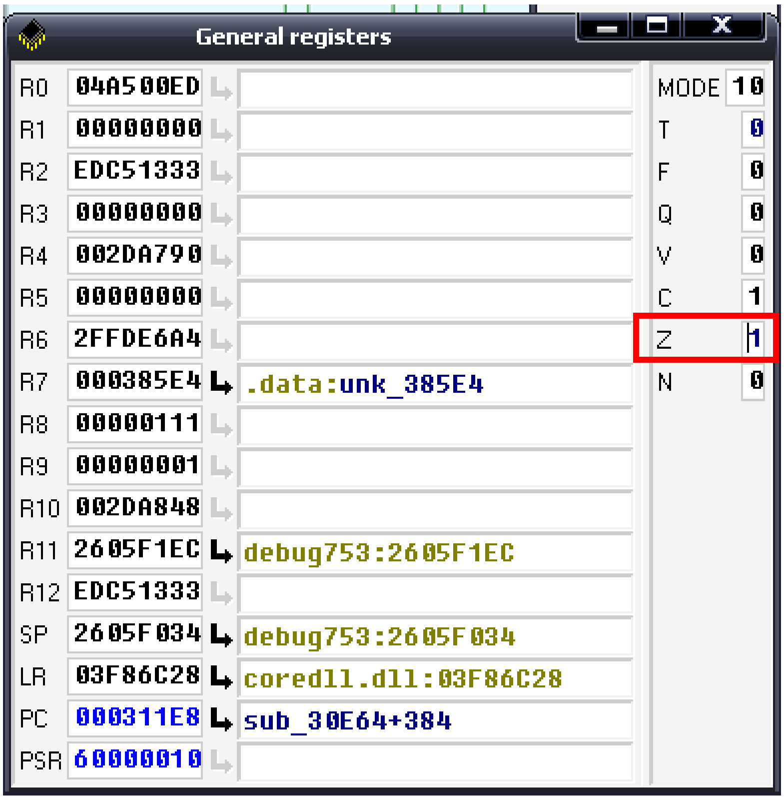
تابع بنفسك.. وستمر على القفزات عند العناوين :

00030FC0	00031190	000311D8



هناك مشكلة في هذه القفزة؟! صحيح أنها لا تؤدي إلى الرسالة مباشرة، ولكن انزل إلى الأسفل وستجد قفزة إجبارية إلى الرسالة، إذن يجب تجاوز هذه القفزة.

وكما تشاهد فهذه القفزة هي BGT أي اقفز إذا كان أكبر تماماً، ويمكن تجاوزها بجعل العلم Z يساوي الواحد (أي يساوي تماماً وليس أكبر تماماً).



عدل قيمة Z كما في الصورة، حسناً تابع التتبع وسترمر على القفزات عند العناوين :

000311F8	00031204

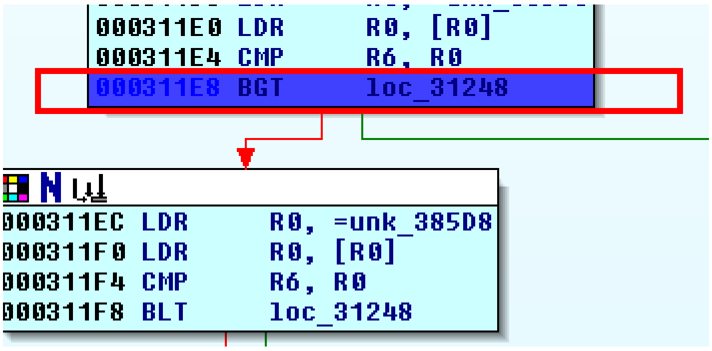
وكلها ستقود للمكان الصحيح، تابع تنفيذ البرنامج بالكامل بالضغط على (F9) :



تم تسجيل اللعبة، والآن سنقوم بعمل Crack للملف ويلزمنا لذلك :

- .Arm Edit Disassembler

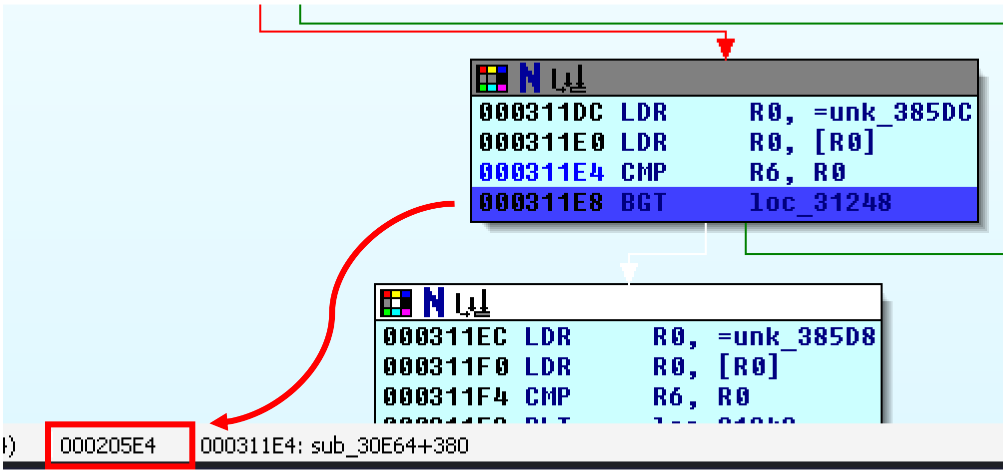
ولكن قبل الأداة لنعد إلى هنا :



الآن سرنعدل :

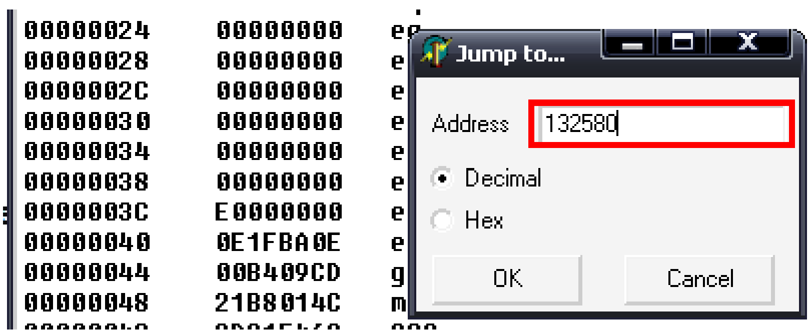
التعليمة القديمة	Hex Code	التعليمة الجديدة	Hex Code
cmp R6,R0	00 00 56 E1	Cmp R6,R6	06 00 56 E1

اضغط على التعليمة التي نريد تعديلها، ولاحظ أن IDA قد حسب الإزاحة في الملف :

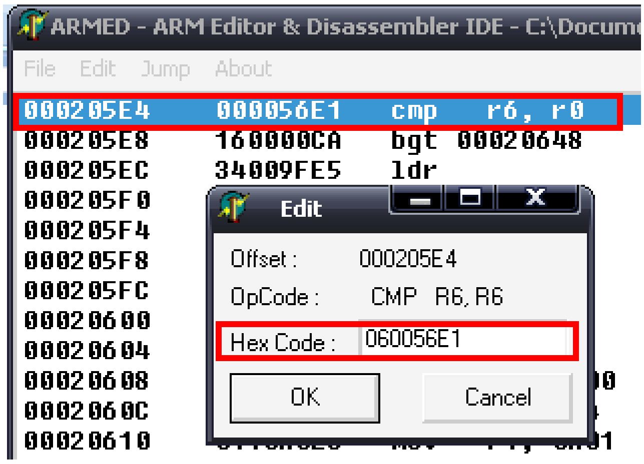


شغل arm_edit_Disassembler :

اختر لعبتنا التي على سطح المكتب File → open → jump → to Offset



OK ثم اضغط بشكلٍ مزدوج على التعليمة المراد تعديلها، وغير التعليمة وكما في الجدول السابق :



File → save ثم OK

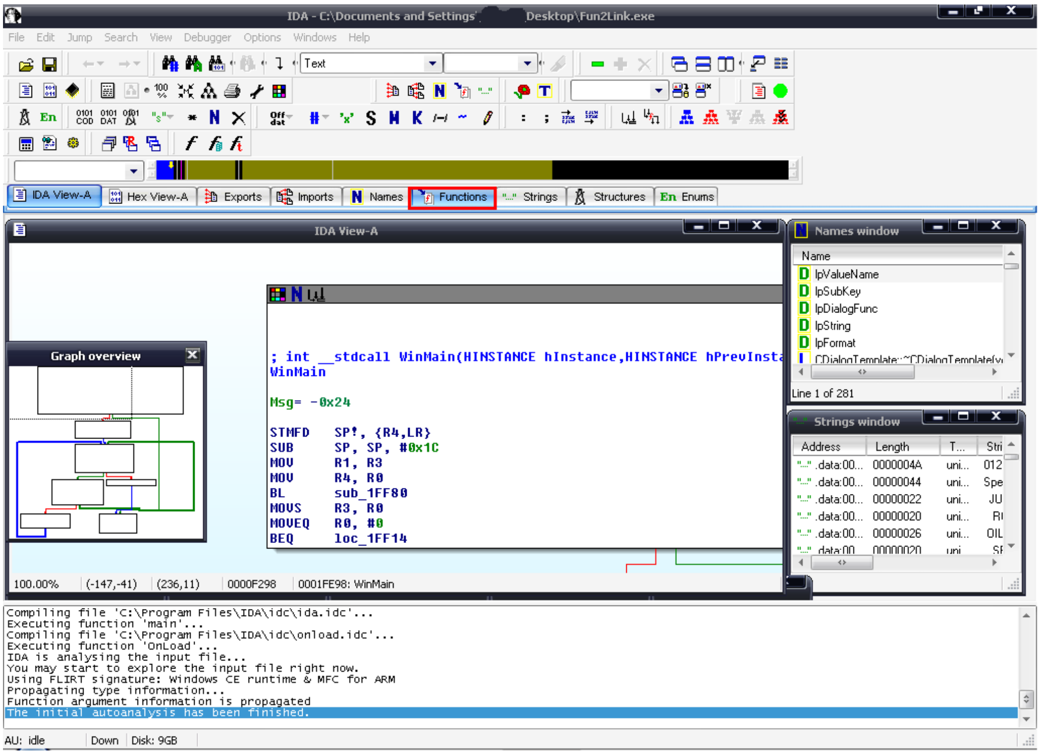
مبارك إنتهينا من الكراك!!

أعد تشغيل الـ Emulator ، حمل اللعبة، استبدل ملف الـ EXE بالكراك، أدخل أي رقم تسجيل، رائع تعمل!!

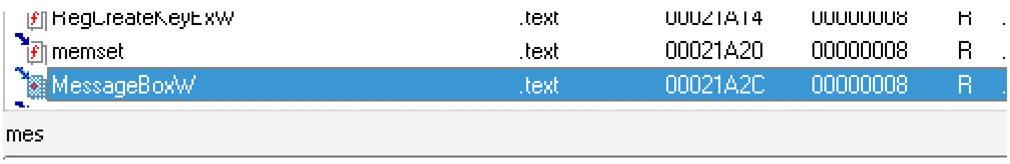
5.2. المثال الثاني : كسر لعبة Fun2Link

Fun2Link (في المرفقات)

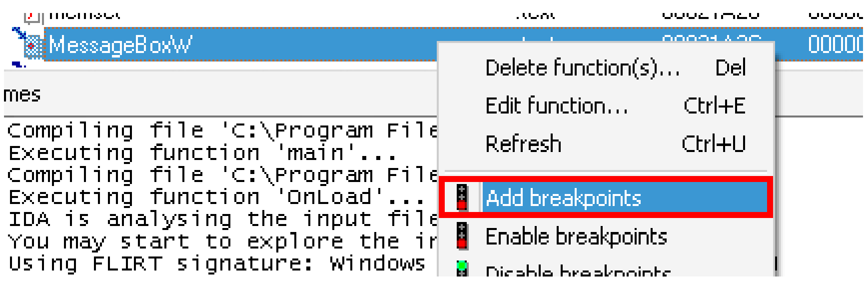
انسخ ملف اللعبة إلى سطح المكتب، ثم اسحبه لداخل برنامج IDA كما فعلنا عند كسر اللعبة السابقة :



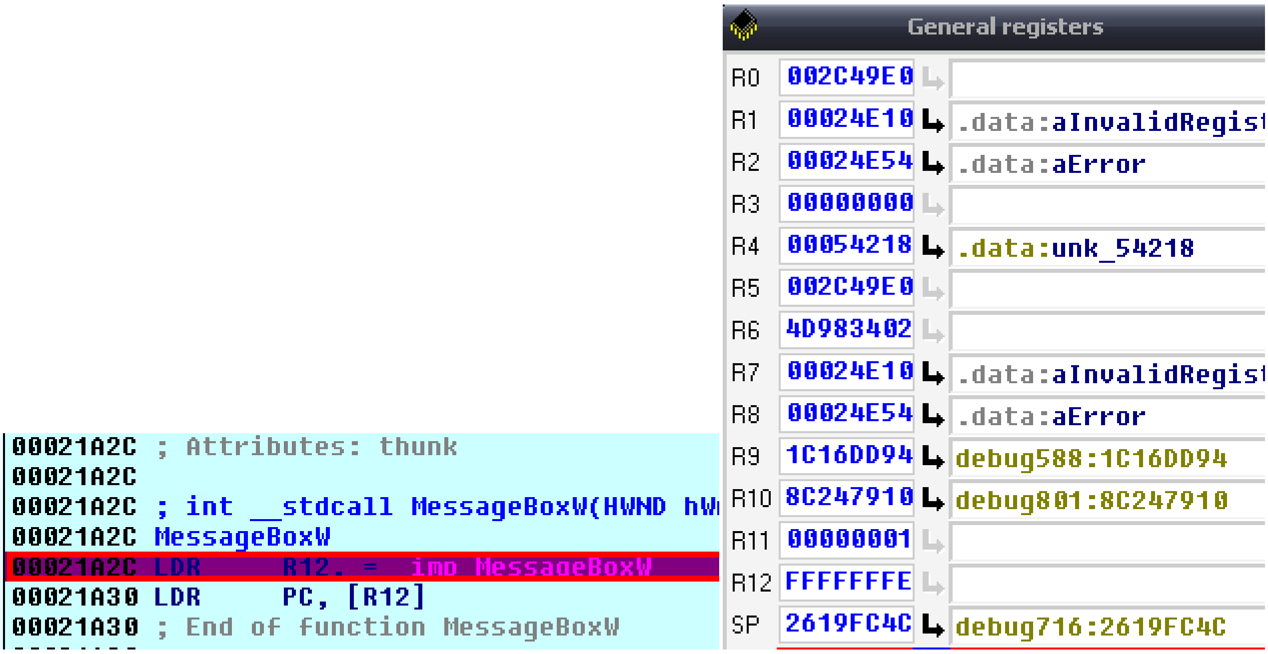
اضغط على Functions وستظهر لك نافذه تحوي جميع الدوال المستدعاة داخل البرنامج، ثم ابدأ بكتابة Messageboxw :



اضغط الزر الأيمن ثم Add break points :

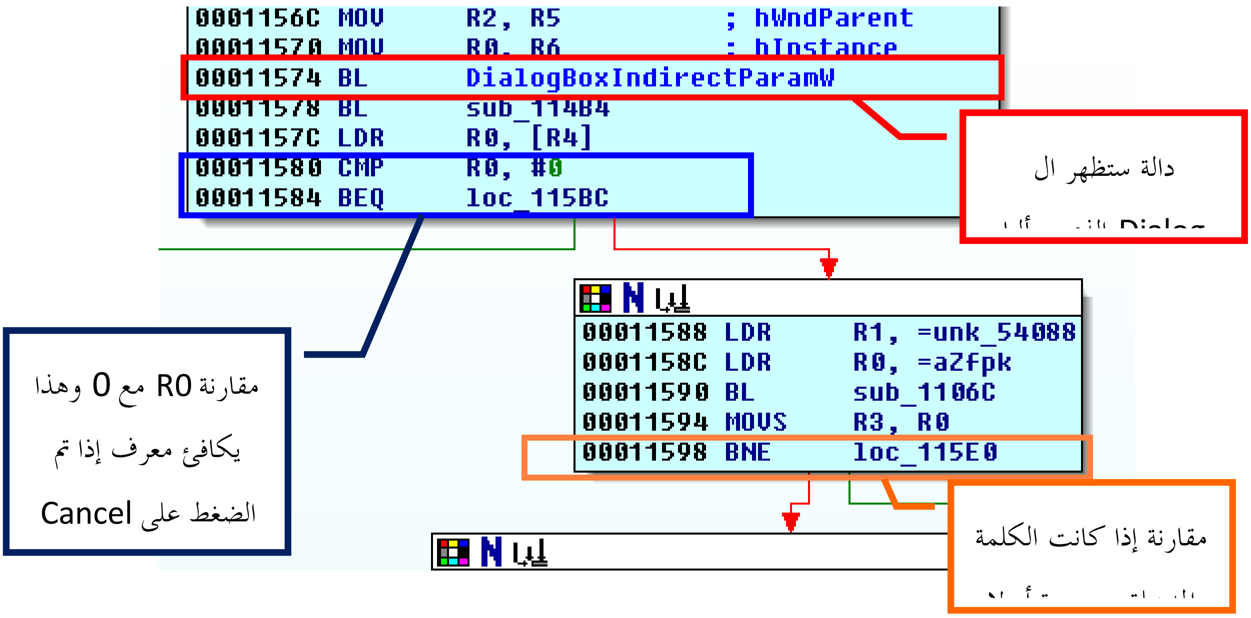


شغل اللعبة، وأدخل أي رقم تسجيل، سيقف البرنامج هنا :



لاحظ المسجل LR فيه عنوان الإجرائية التي استدعت دالة messageboxW !!

هل تشاهد المربع الأزرق؟ انقر على السهم الموجود داخله لكي نذهب إلى هذه الإجرائية، أنت هنا :



أضف نقطة توقف عند القفزة البرتقالية، وعند الوصول إليها اعكس قيمة المسجل Z ، تا بع التنفيذ:

تهانينا الآن استمتع باللعبة

تم بعونه تعالى

الباب السابع: تطبيقات عملية موسعة

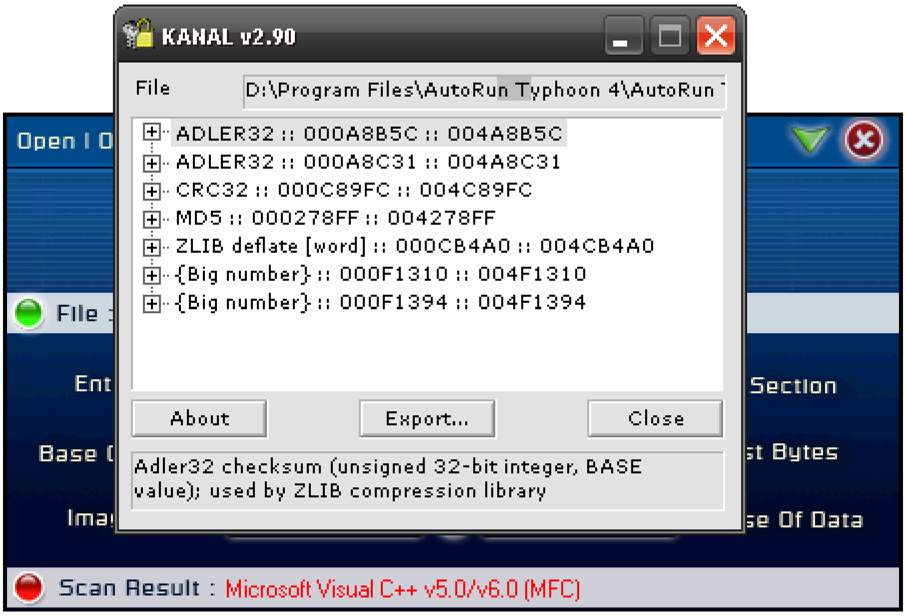
شرح كسر برنامج AutoRun Typhoon Pro v 4.0.4

- OllyDbg.
- AT4RE FastScanner v2.0 .
- ExeInfo PE v 0.0.1.9.

تجد الضحية في المرفقات (التحميل من مرفقات الكتاب).

مثالنا في هذا الدرس سيتطرق إلى العديد من الأمور في كسر البرامج، أهمها منع اتصال البرنامج بسيرفر الشركة للتحقق من التسجيل، وأمور أخرى سنتطرق أليها أثناء الدرس.

قبل كل شيء سنقوم بفحص البرنامج باستخدام AT4RE FastScanner v2.0 وهي خطوة مهمة تعطينا فكرة أوسع عن البرنامج مما يسهل علينا كسره :

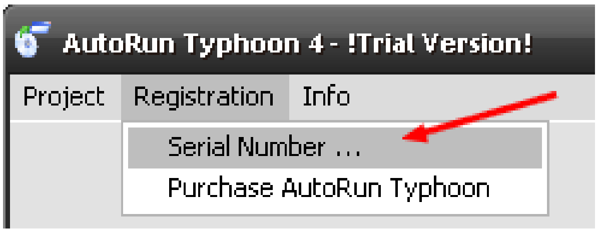


البرنامج يستخدم العديد من مكاتب التشفير، سندرس عمل البعض منها لاحقاً.

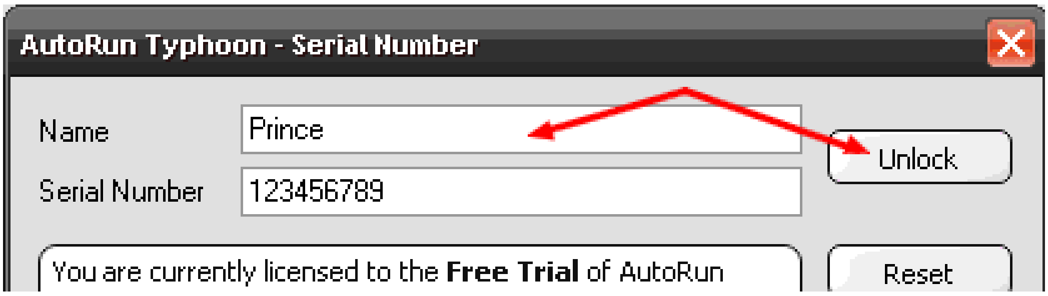
قم بتشغيل البرنامج لنتعرف عليه، ولكي نأخذ عنه فكرة أشمل.

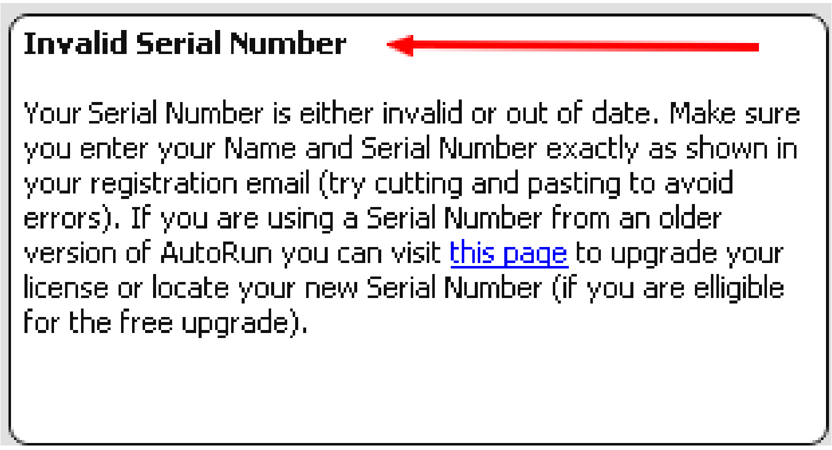


لا نجد أي رسائل تطالبنا بالتسجيل، ولكن سنقوم بعملية التسجيل بأنفسنا :



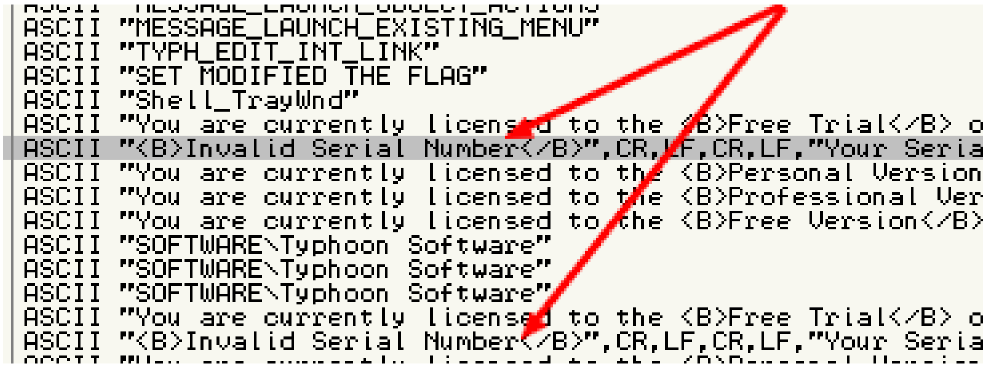
أدخل أي اسم وأي رقم، ثم اضغط الزر unlock :



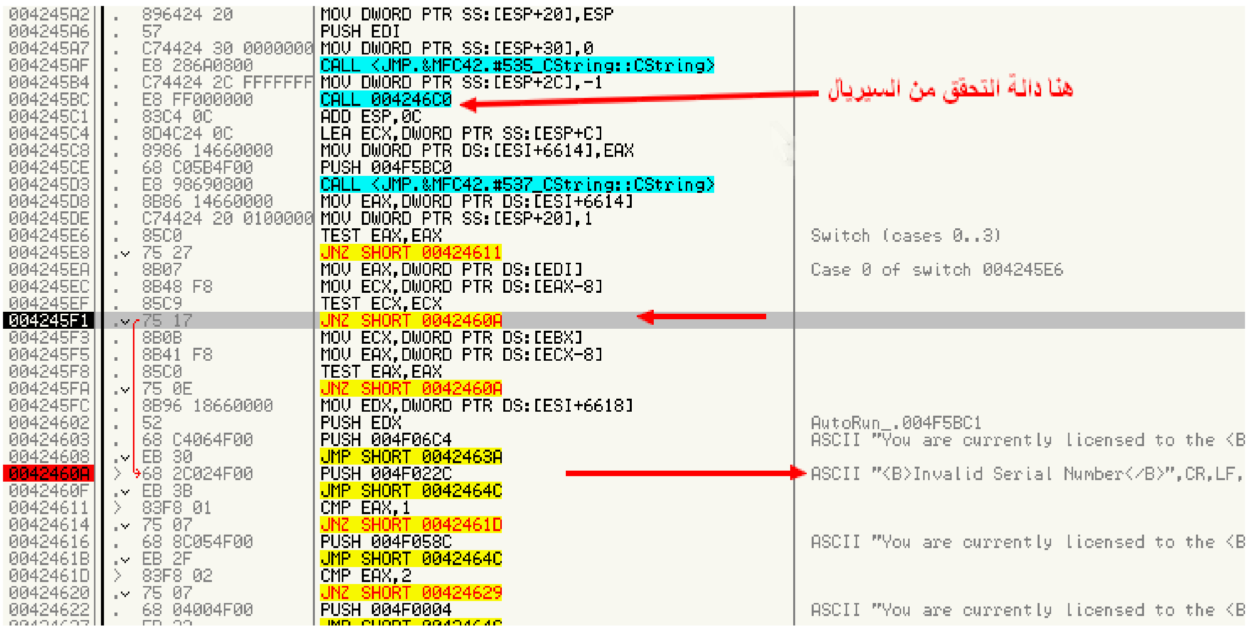


نلاحظ أن البرنامج أصم؛ فهو يتحدث بلغة النصوص وبدون رسائل !!

لكن أخذنا فكرة بسيطة عن البرنامج، وما سنقوم به هو البحث عن النص الذي عرضه البرنامج، وهو Invalid Serial Number في برنامج OllyDbg :

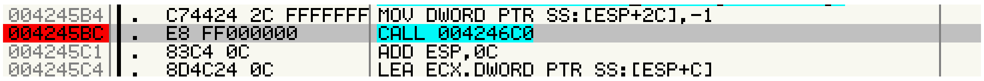


أكيد أنك تعرف ما عليك عمله؟! وضع نقاط توقف في كلا الموضعين، ثم شغل البرنامج واذهب إلى واجهة التسجيل وأكتب أي أسم ورقم، ثم أضغط موافق، ستجد أنه توقف عند أحدهما :

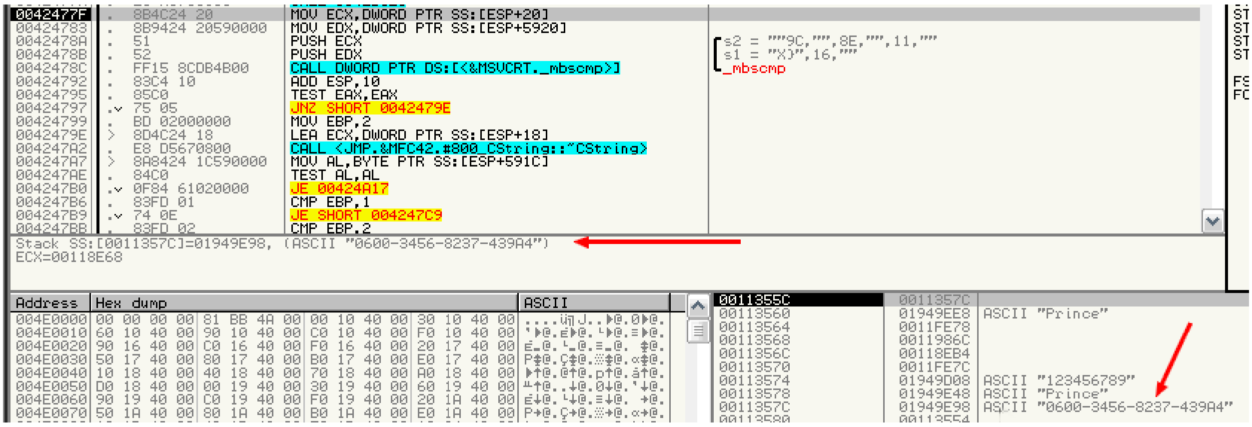


ستجد أن هناك قفزات مشروطة تسبقها دالة، بالتأكيد هي التي تتحقق من السريال، هل هو صحيح أم لا؟ وفي حالتنا سنذهب إلى رسالة الخطأ، وما ستقوم به هو عمل نقطة توقف عند دالة التحقق عند العنوان : 004245BC

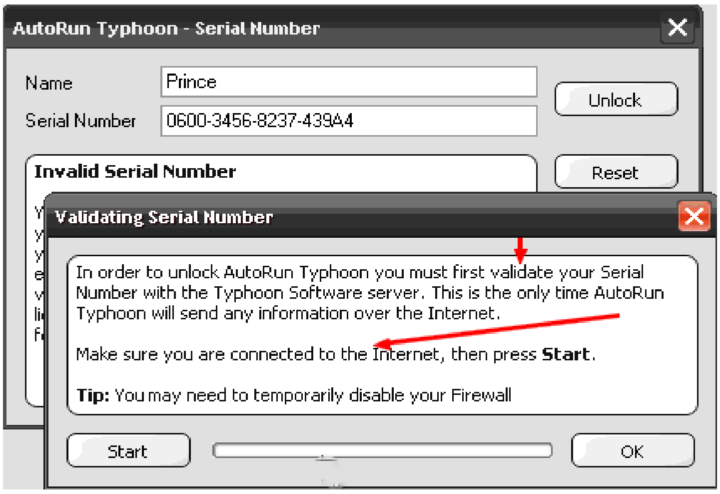
شغل البرنامج الآن، وأدخل أي بيانات، ثم اضغط موافق، وعند دالة التحقق اضغط الزر F7 للذهاب إليها :



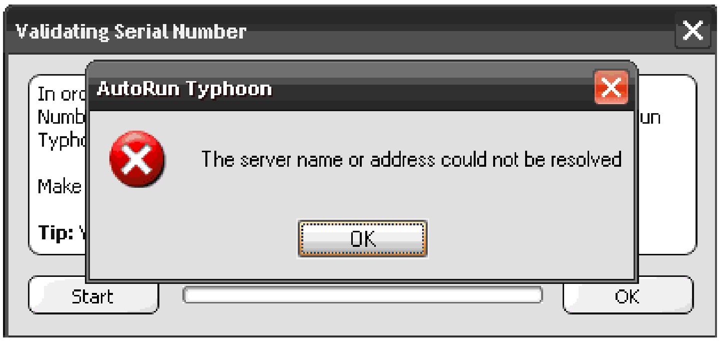
الآن تتبع البرنامج، وافتح عينيك على نافذة المسجلات والمكدس :



وجدنا السريال وبدون أي عناء، الآن قم بنسخة وشغل البرنامج وأدخل السريال الصحيح الذي وجدناه في البرنامج، ثم اضغط موافق :

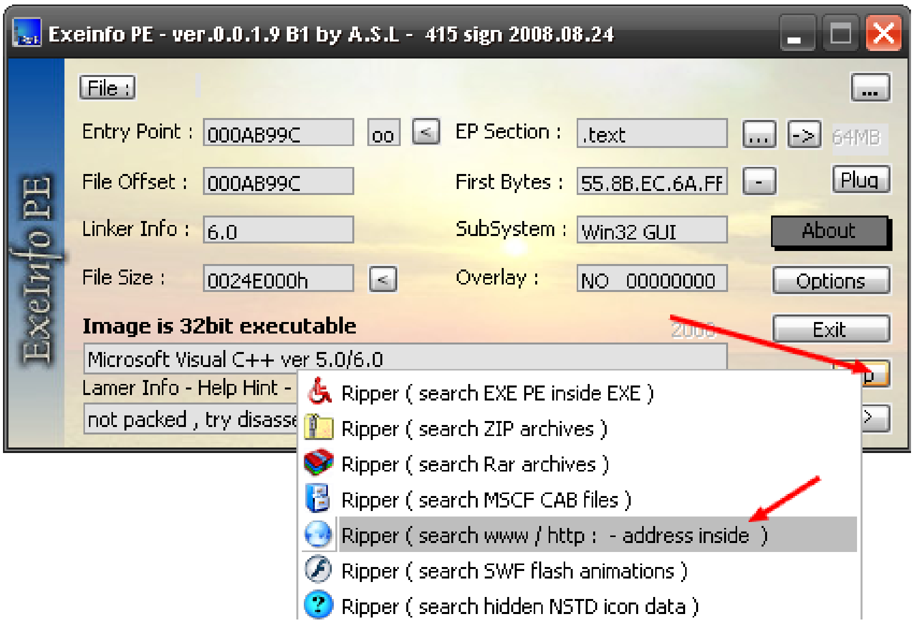


نلاحظ أن السريال صحيح بدليل ظهور هذه الرسالة، وهو يريد منا ربط الانترنت، ليتحقق من التسجيل لدى سيرفر الشركة، وهناك سيتأكد من أن بيانات التسجيل التي أدخلناها غير موثوقة لديه، وأننا حصلنا عليها بطريقة غير شرعية ولن يتم التسجيل في البرنامج !

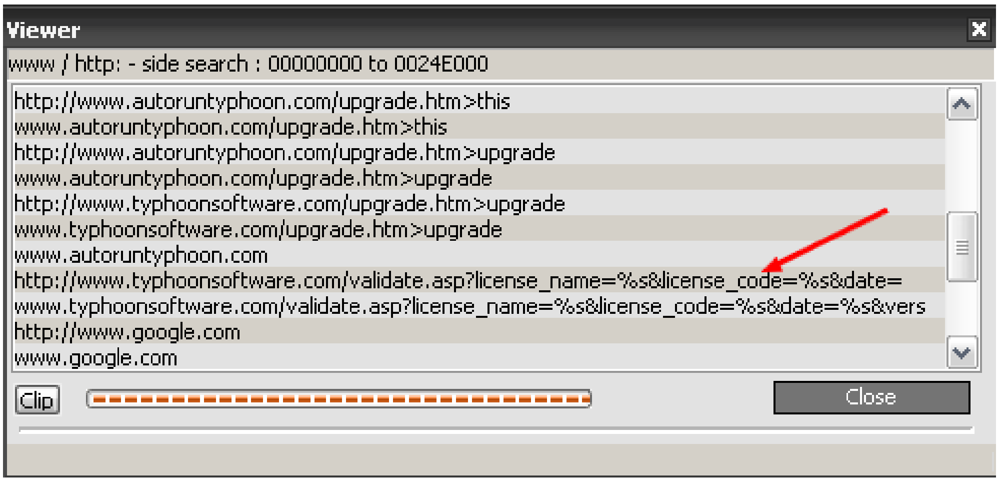


ستظهر لنا هذه الرسالة عند الضغط على الزر start لذا يجب علينا الآن إكمال عملية التسجيل، وتخطي أو منع البرنامج من التحقق من التسجيل عبر الإنترنت. في الحقيقة لن نتتبع مصدر هذه الرسالة، ولكن ما سنقوم به هو الوصول إلى الدوال التي يقوم من خلالها البرنامج بالاتصال عبر الإنترنت للتحقق من التسجيل.

حمل البرنامج في ExeInfo PE v 0.0.1.9 :



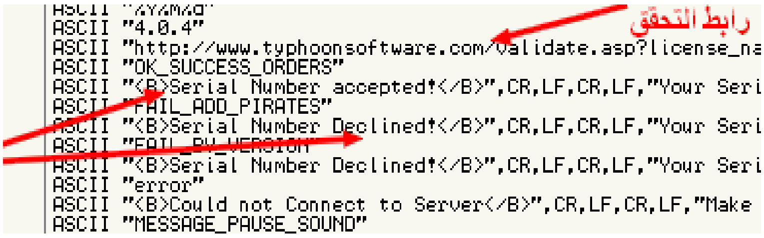
سنقوم من خلال هذا البرنامج البحث عن جميع الروابط الموجودة في ضحيتنا :



ستجد العديد من الروابط وما يهمنا هو الموقع الخاص بالتحقق من التسجيل, وهو المشار إليه بالسهم.

بمجرد الضغط على الرابط سيتم نسخه تلقائياً.

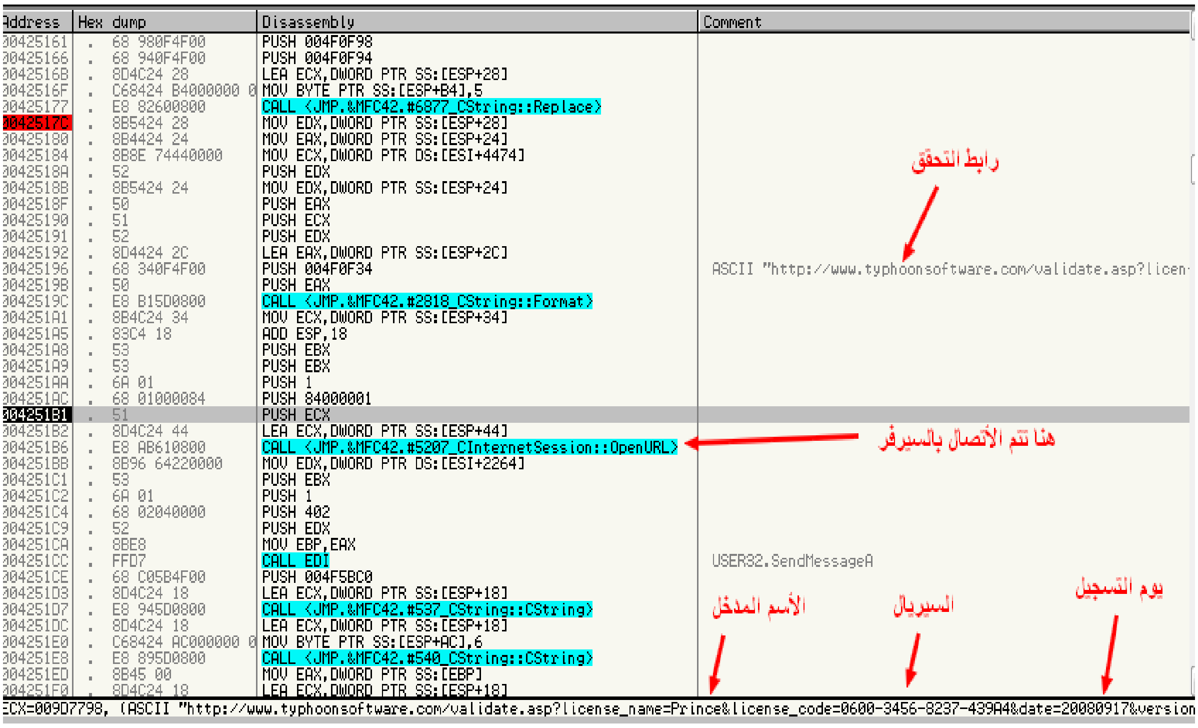
ابحث عن هذا النص أو عن هذا الرابط في OllyDbg :



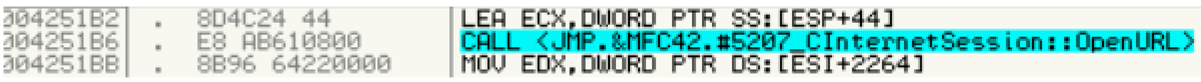
نلاحظ وجود الرابط وبالأسفل منه عبارتي: القبول ورفض السريال، وأيضاً عبارة عدم الاتصال بالسيرفر.

إذن في هذا المكان من البرنامج تتم عملية الاتصال بالسيرفر والتحقق من التسجيل، هل هو صحيح أم مزور؟ وهنا بالتحديد سنقوم بإلغاء أو تخطي هذه العملية بجعله يذهب إلى رسالة الصواب.

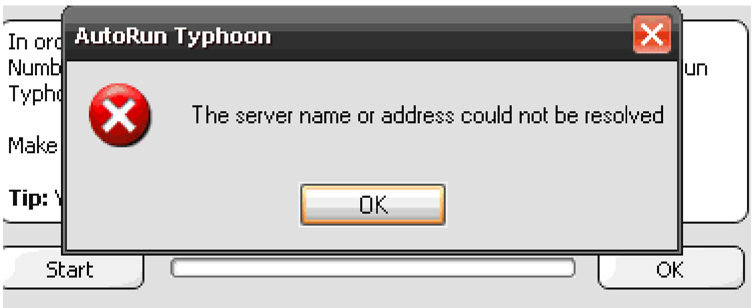
اضغط بشكل مزدوج على النص المؤشر لندرس كيفية يتم تخطي عملية اتصال البرنامج بالسيرفر عبر الإنترنت:



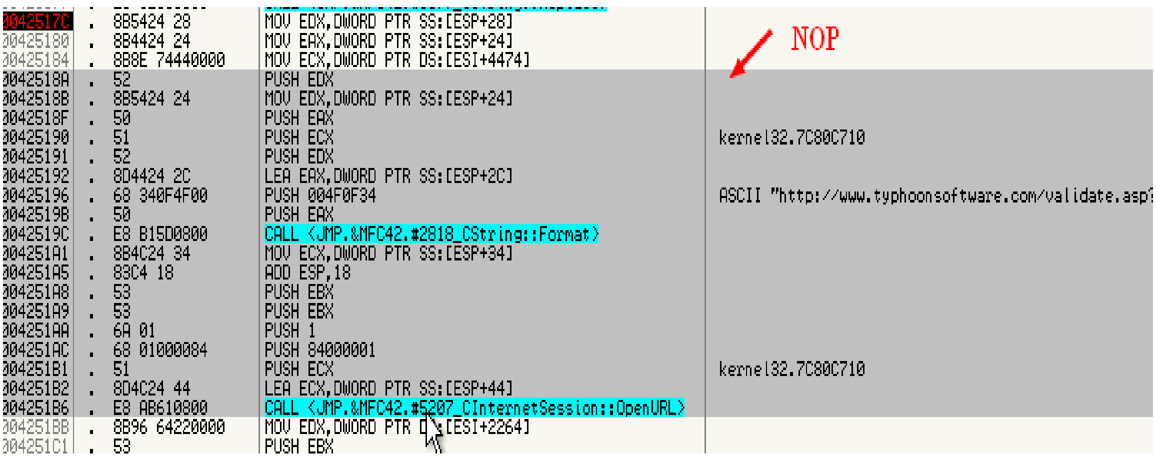
ضع نقطة توقف عند العنوان 0042517C وأزل جميع النقاط السابقة التي وضعناها، ثم اضغط على الزر Start مرة أخرى وسيتوقف عند تلك النقطة، تتبع وستلاحظ أن البرنامج يقوم بإرسال البيانات التي أدخلناها إلى سيرفر الشركة عبر ذلك الرابط :



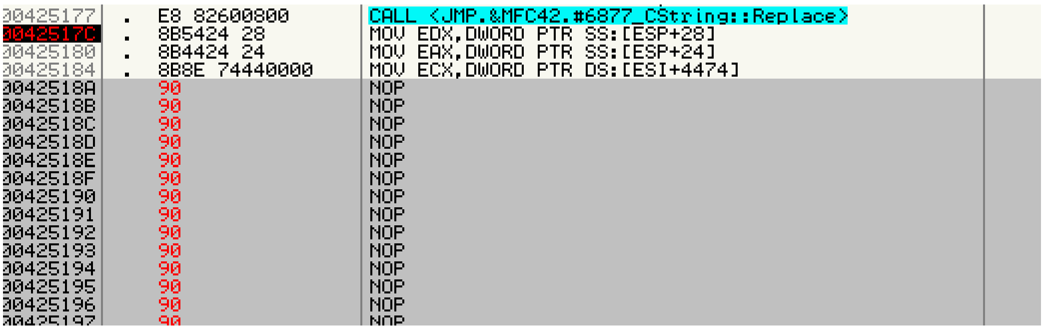
وعندما تصل إلى هذا العنوان 004251B6 :



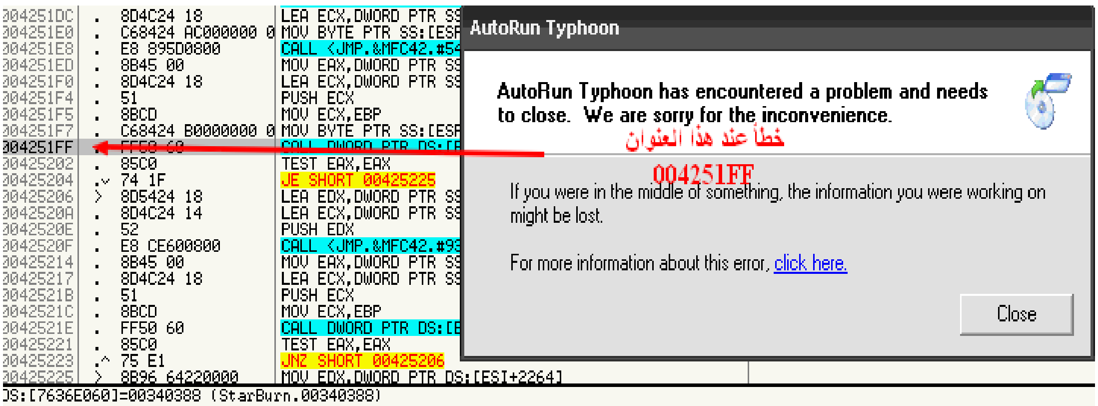
ستظهر هذه الرسالة، وهي رسالة الخطأ؛ لعدم وجود اتصال بالسيرفر :



ما سنقوم بعمله هو : استبدال NOP بالتعليمات التي تتهيأ بعملية الاتصال وإرسال بيانات التسجيل إلى السيرفر، وتكون عملية الاستبدال بدءاً من العنوان 0042518A وإلى العنوان الذي تظهر فيه رسالة الخطأ، أي عند هذا العنوان 004251B6 :

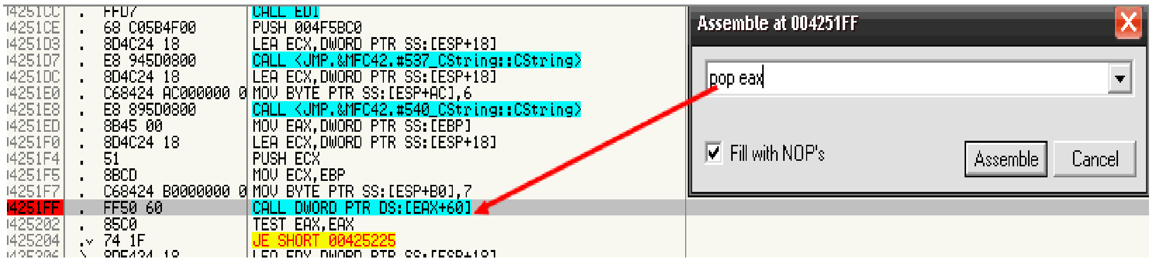


الآن شغل البرنامج واضغط على الزر Start لنشاهد نتيجة ما قمنا به :

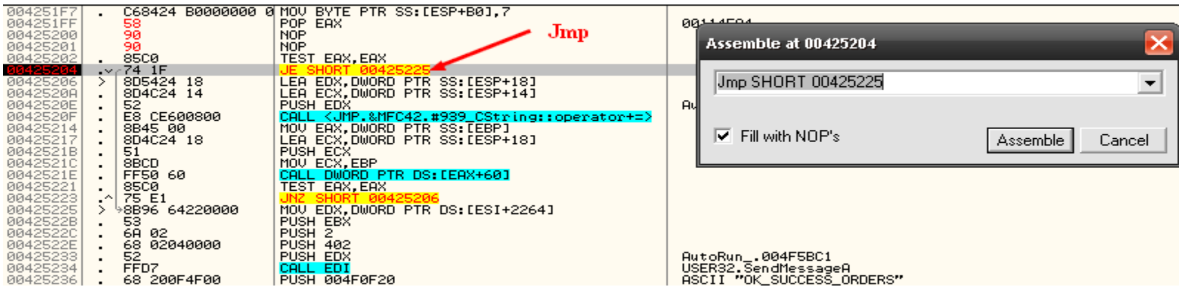


ستجد حدوث خطأ عند ذلك العنوان الموضح بالصورة، وسبب ذلك هو ما قمنا به من حذف التعليمات، وسيتم إغلاق البرنامج بعد حدوث ذلك الخطأ !!

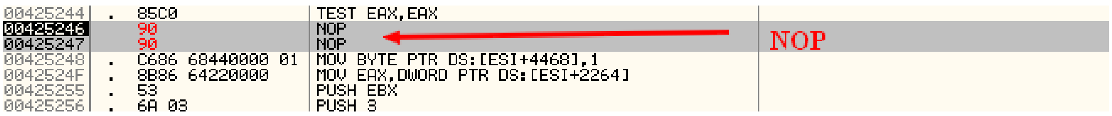
لا بأس أعد تشغيل البرنامج واذهب إلى نقطة التوقف في المكان الذي يتحقق فيه البرنامج من التسجيل، واستبدل NOP بتلك التعليمات، وضع نقطة توقف عند العنوان الذي حدث فيه خطأ كعلامة لك، ومن ثم أدخل البيانات الصحيحة وتتبع البرنامج إلى أن تصل إلى العنوان الذي حدث فيه الخطأ :



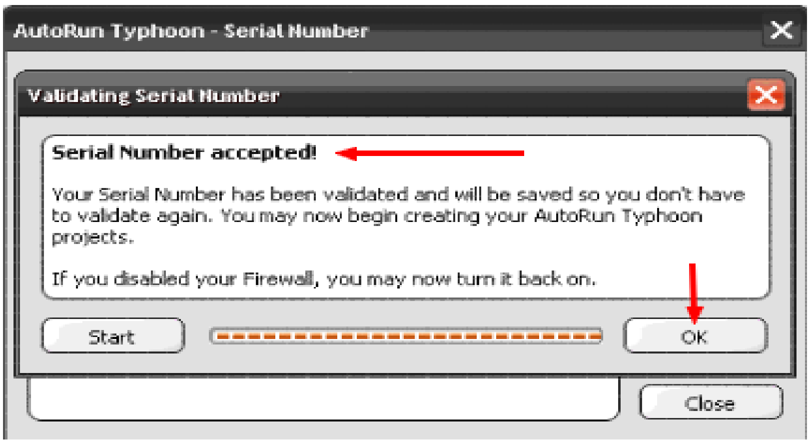
الآن نستبدل Pop EAX بتلك التعليمة حتى لا يحدث خطأ في التعليمة التي تليها TEST EAX,EAX :



ثم نقوم بتعديل القفزة المشروطة إلى JMP SHORT 00425225 :

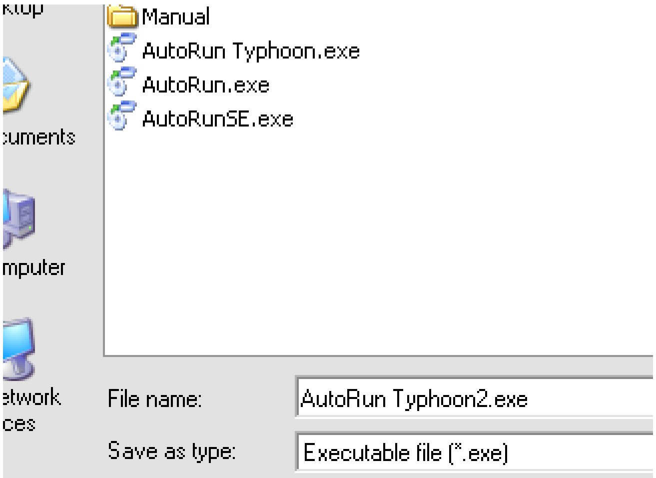


وأيضاً هذه القفزة إلى NOP حتى تمر من رسالة الصواب، ويتم الانتهاء من التسجيل في البرنامج، ويتم قبول السريال :

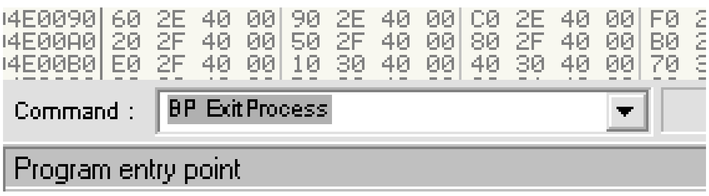


وبهذه العملية جعلنا البرنامج يقبل بيانات التسجيل بدون عملية التحقق من التسجيل عبر سيرفر الشركة. و بذلك نكون قد انتهينا من عملية التسجيل في البرنامج بنجاح.

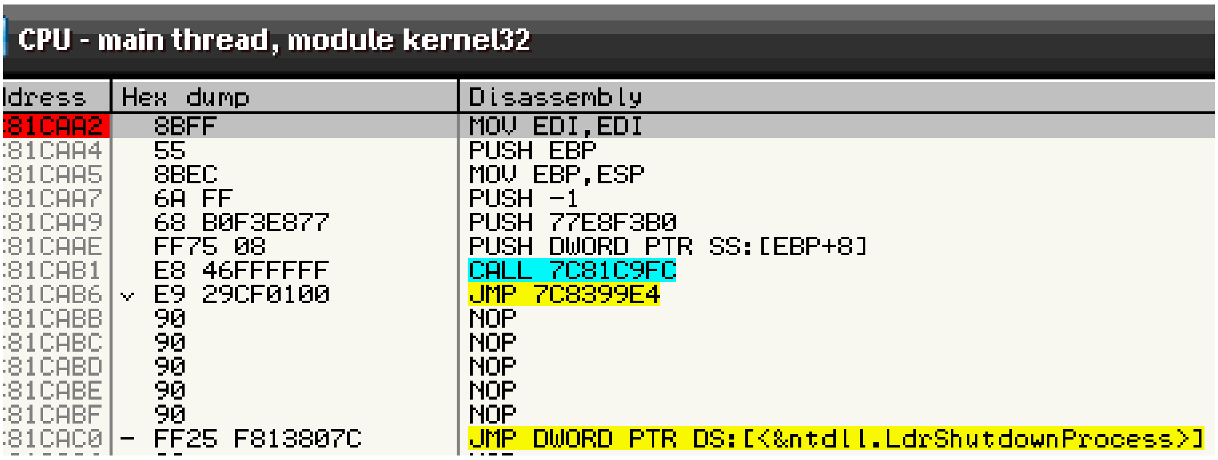
> توجد برامج تتحقق من التسجيل عند تشغيلها أثناء اتصالك بالإنترنت بدون علمك، وتجد أنها تلغي بيانات التسجيل. هذه البرامج بعضها أمره سهل وما عليك سوى تحديد الروابط في البرنامج؛ لتعرف أيًّا منها التي تقوم بعملية الاتصال بالسيرفر، وإن لم تستطع تحديد الرابط الصحيح فضع نقاط توقف على جميع الروابط، وعند الوقوف على أحدها قم بتغيير أي حروف من ذلك الرابط، أي قم بتغيير حرف من :
>
> www.typhoon.com ليصبح : www. typhaon.com
>
> أي إلى موقع مجهول؛ حتى نمنع اتصال البرنامج بالسيرفر.
>
> لكن هذه الطرية لا تنفع في كل الحالات.



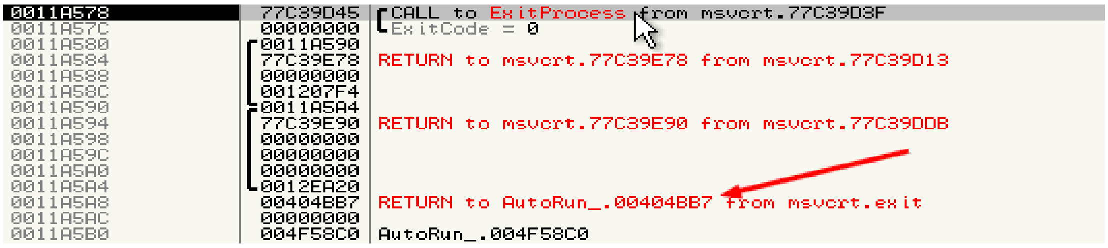
نحفظ التغيرات التي قمنا بها، ثم نشغل البرنامج، وعندها ستجد أنه لا يعمل نهائياً، وهنا نعرف أن البرنامج يستخدم CRC32 وهي خوارزمية تستخدم لكشف أي تعديل في البرنامج، وقد استخدمها المبرمج ليمنع أي تعديل بالبرنامج، ويبدو أنه ذكي قليلاً؛ فهو لم يظهر لنا أي رسالة تبين بأن البرنامج ملعوب به، والتي كانت ستسهل علينا كثيراً في إيجاد هذه الحماية وتخطيها :



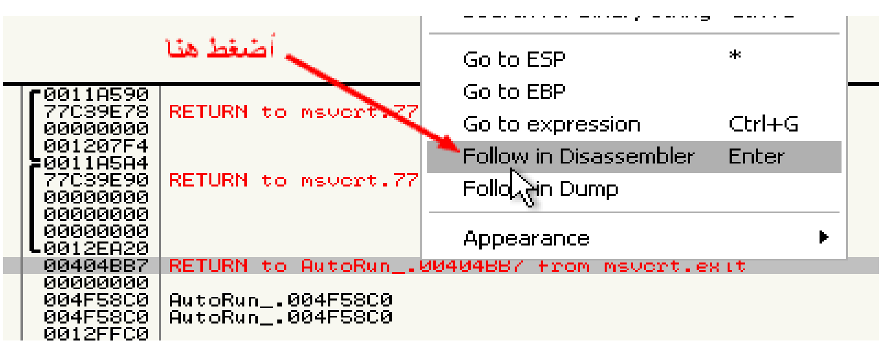
لكن لا عليك فقط حمل الملف المعدل في olly وفي شريط الأوامر أدخل هذا الأمر BP ExitProcess وما قمنا به هو نقطة توقف على هذه الدالة لنعرف مصدر الدالة التي يستدعيها البرنامج عند إغلاقه :



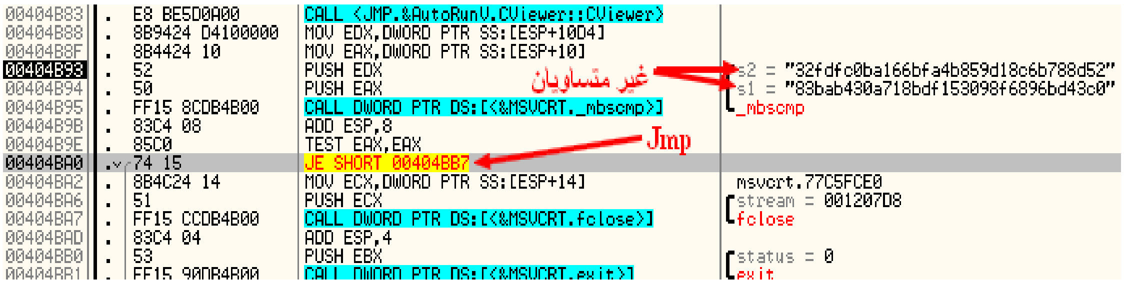
الآن شغل البرنامج، ستجد أنه توقف في المكان الموضح في الصورة التي أمامك، وهنا قم بإزالة هذه النقطة :



توجه إلى المكدس في برنامج OllyDbg والذي سيبين لنا الدالة التي تستدعيها من خلال البرنامج، وهي عند هذا العنوان في البرنامج : AutoRun_.00404BB7



نذهب إليها ونرى ماذا يوجد لدينا في ذلك العنوان ؟



إذن العملية واضحة.بما يقوم به البرنامج من عملية حسابية بين رقمين، وغالب الظن أنها بين قيمة مخزنة في البرنامج – والذي وضعها المبرمج– مع القيمة الثانية التي ولدتها حماية CRC32 في الملف الذي قمنا بالتعديل فيه، والنتيجة: القيمتان غير متساويتين، وهي ما ستجعل تلك القفزة المشروطة لا تقفز، وإنما ستمر إلى الدوال التي تقوم بإغلاق البرنامج.

وهنا تعرف ماذا يجب أن تفعل؟ وهو تغير القفزة المشروطة إلى قفزة غير مشروطة، أي إلى Jmp احفظ التغيرات التي قمنا بها، وشغل الملف ستجده يعمل وبدون أي مشاكل، وبهذا نكون قد انتهينا من كسر البرنامج بشكل نهائي.

ستلاحظ عند تشغيل ملف البرنامج الغير معدل أنه مسجلٌ باسمنا، فهو لا يقوم بالتحقق من السيرفر سوى مرة واحدة، وهي عند إدخال بيانات التسجيل.	

إيجاد سريال وكتابة كيجن لبرنامج AVI Video Converter [DES]

Keygenning AVI Video Converter (DES Protected)

سنقدم في هذا الدرس شرحاً لطريقة إيجاد السريال و كتابة كيجن لبرنامج مشفر بخوارزمية التشفير DES

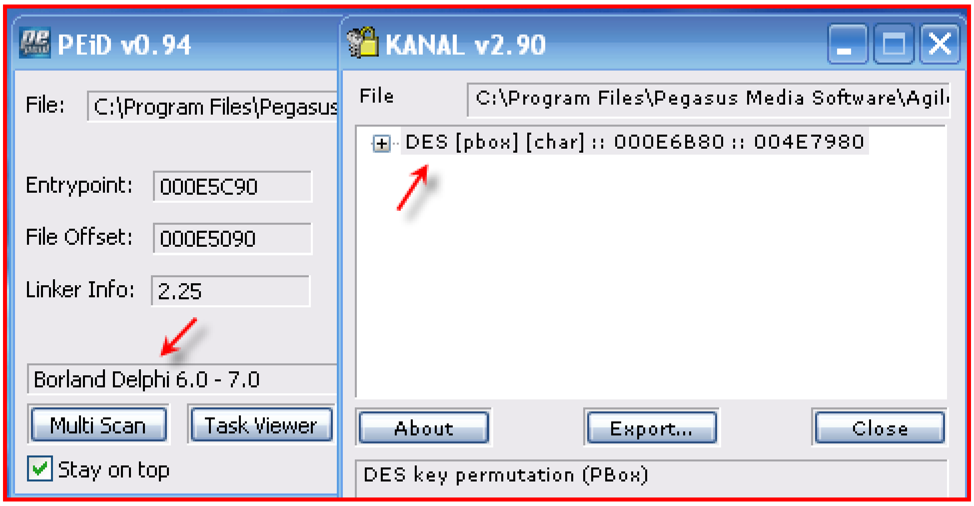
- OllyDbg.
- SnD Reverser Tool.
- Delphi 7.

تجد الضحية في المرفقات باسم AVI Video Converter 2.1.3

(التحميل من مرفقات الكتاب).

كيف عرفنا أن البرنامج يستخدم خوارزمية DES ؟

ببساطة بفحص البرنامج ببرنامج PEiD واستخدام البلاغين Kanal (للكشف عن التشفير) :



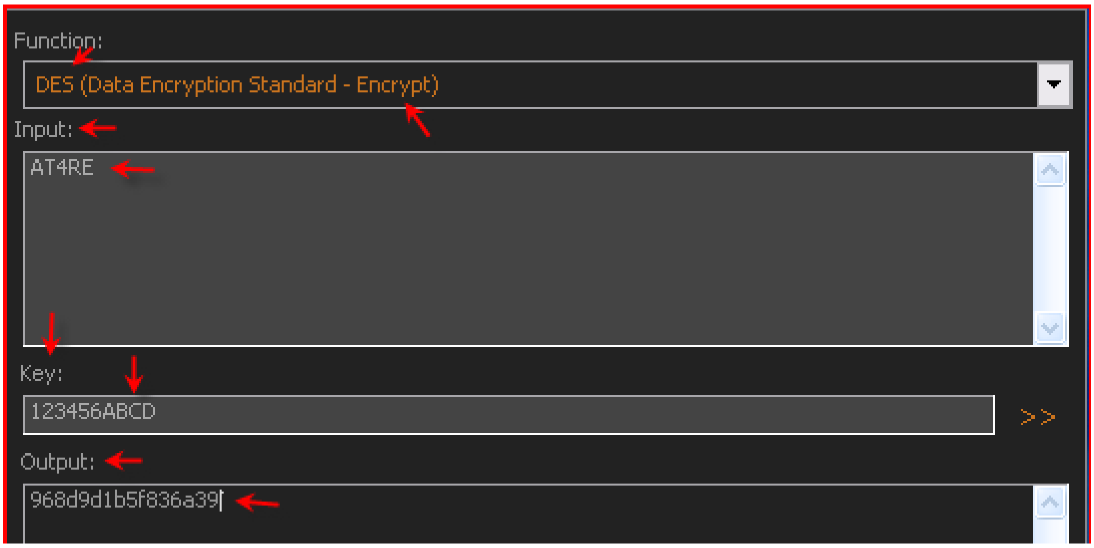
ماذا استفدنا من معرفة أن البرنامج يستخدم خوارزمية التشفير DES ؟

- لأنه قد يستخدمها البرنامج أثناء حسابه للسريال أو بيانات التسجيل (وهذا ما يحدث غالباً).

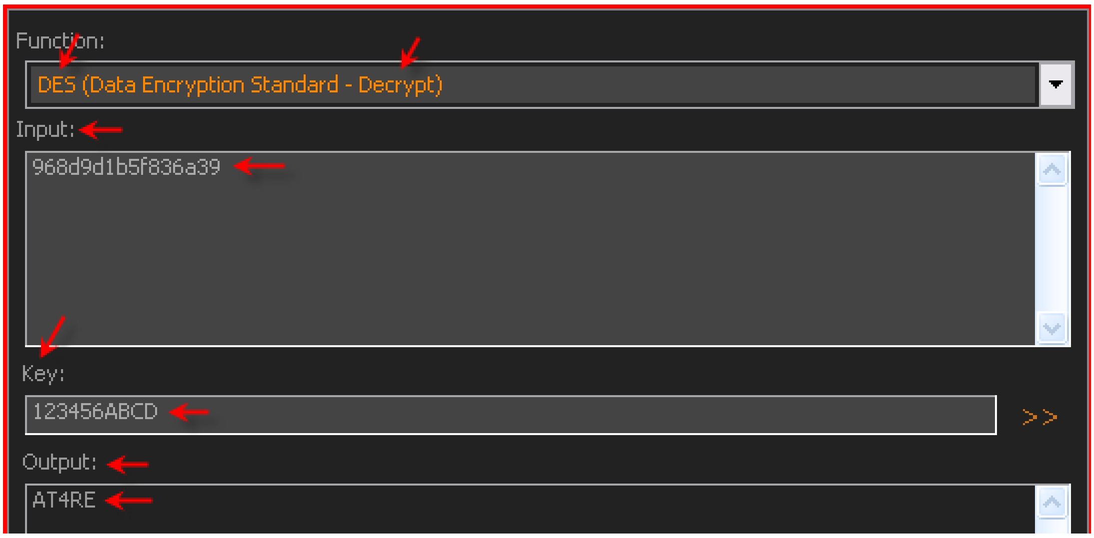
- ينبهنا للبحث عن أمور تتعلق بالخوارزمية المستخدمة داخل البرنامج، فهنا لدينا DES .
- أولاَ : تستخدم طريقين : تشفير EnCrypt وفك التشفير DeCrypt .
- ثانياً: هي بحاجة إلى مفتاح Key تستخدمه في عمليات التشفير وفك التشفير، وهو معلن، أي يجب أن يتواجد داخل البرنامج أو يتم حسابه داخل البرنامج.

فعلى سبيل المثال: إذا كان لدينا الكلمة AT4RE وأردنا تشفيرها بــ DES باستخدام المفتاح: 123456ABCD .

سنستخدم الأداة Snd Reverser Tool ، اختر DES {EnCrypt} ثم أدخل البيانات العلوية، ستجد أن الرسالة المشفرة هي: 968d9d1b5f836a39 ولاحظ أنها بالنظام الستة عشري (HEX) :



الآن خذ القيمة : 968d9d1b5f836a39 وضعها في Input وأبقي نفس المفتاح، ولكن اختر Des .. Decrypt ستلاحظ أن فك التشفير أنتج لك AT4RE ولكن انتبه إلى أن المفتاح بقي نفسه للعمليتين :



إذن وكما تلاحظ العملية غاية في السهولة مع معرفة المفتاح المستخدم في خوارزمية التشفير.

I. أولاً: إيجاد السريال

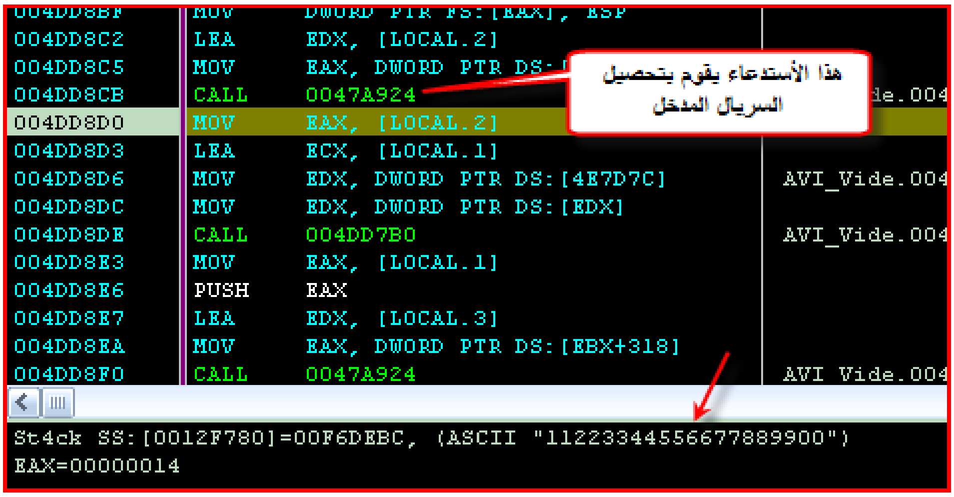
– افتح البرنامج وأدخل أي بيانات تسجيل واضغط Register ستظهر لك العبارة Registration is failed. Please check .

– حمل البرنامج في OllyDbg وابحث عن الرسالة السابقة، وضع نقطة توقف عند بداية الإجراء 004DD8A0 .

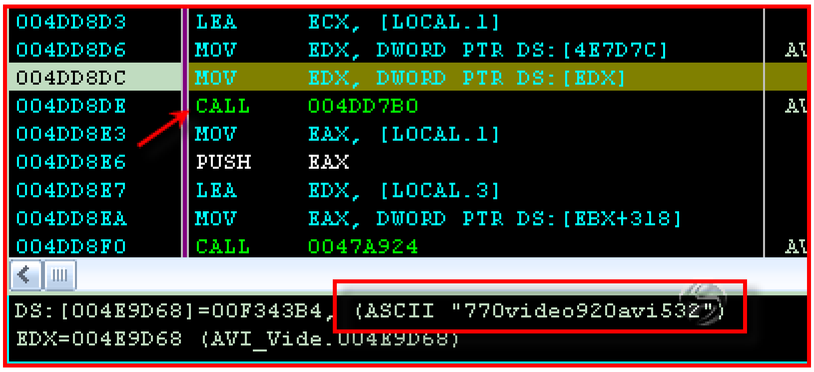
– شغل البرنامج وتوجه إلى التسجيل وأدخل البيانات التالية :

- الاسم : Roben
- الرقم : 11223344556677889900

– اضغط Register وسيتوقف المنقح عند نقطة التوقف التي وضعناها قبل قليل. ماذا يفعل البرنامج ببيانات التسجيل الخاصة بنا؟ لنتتبع أولاً بالضغط على F8 :

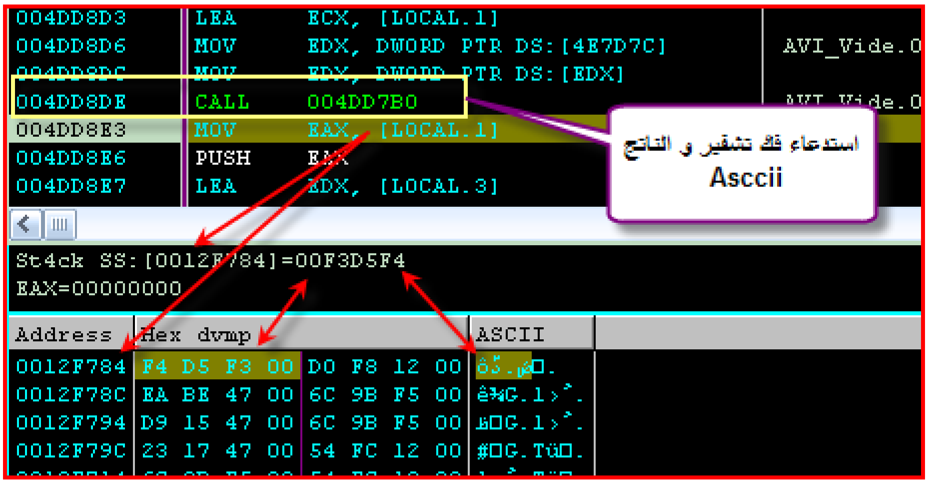


– نتابع الضغط فتظهر لنا سلسلة نصية مميزة وهي : 770video920avi532 لا بد أن يكون هذا هو مفتاح التشفير، والاستدعاء التالي له يقوم بعملية تشفير أو فك التشفير، لنلاحظ :

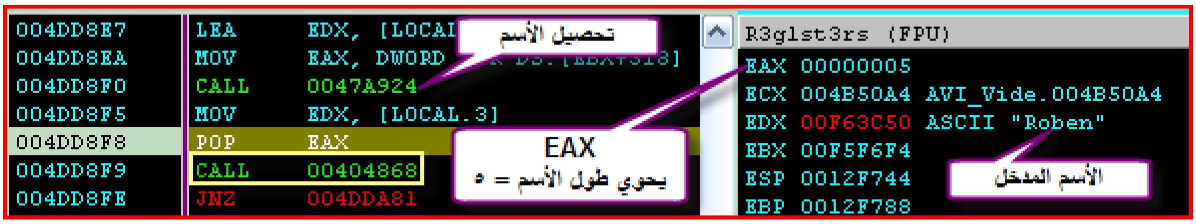


– إذا كان الناتج التالي للاستدعاء عبارة عن سلسلة ستة عشريّ إذن فالعملية المجراة هي تشفير، أما إذا كان الناتج سلسلة نصية (تشمل جميع الـ Char ، على سبيل المثال : AsD9□ófÁ) ولكن OllyDbg لا يظهرها إذا احتوت على رموز، بل يظهر مقابلها في الستة عشري (Hex)) فالعملية المجراة فك تشفير.

– لا حظ ناتج الاستدعاء Call 004DD7B0 وسيتم وضعه في EAX ولنحاول حفظه جيداً 00F3D5F4



– لنتابع أيضاً :



– الآن سيقوم البرنامج بمقارنة الاسم المدخل مع قيمة معينة من خلال الاستدعاء Call 00404868 . لندخل إلى داخل الإجراء ونراقب المقارنة.

– نلاحظ وجود مقارنة بين الاسم المدخل Roben والقيمة الناتجة عن فك التشفير للسريال، وهي : 00F3D5F4



– البرنامج يقوم بعملية عكسية وفق الخطوات التالية :

1) يتم أخذ السريال، ويقوم بفك تشفيره بـــ DES مستخدماً المفتاح : 770video920avi532 .

2) يقارن ناتج فك التشفير مع الاسم المدخل Roben .

إذن كيف يتم إيجاد السريال الصحيح للاسم المدخل ؟

الأمر بسيط !! سنأخذ الاسم ونشفره بـــ Des باستخدام المفتاح وناتج التشفير سيكون السريال؛ لأن البرنامج عندما سيفك تشفير هذا السريال سينتج معه الاسم الذي شفرناه.

– لنستخدم الأداة : SnD Reverser Tool لإيجاد سريال صحيح للاسم Roben مستخدمين :

Key = 770video920avi532 -------- Input = Roben ----- Function = DES .... EnCrypt



-- ثمانينا البرنامج مسجل :



2. ثانياً: كتابة كيجن للبرنامج

- لابد بداية أن تتوفر لك المكتبة DES لتستطيع استدعاء التشفير وفك التشفير.

- سنستخدم في كتابة الكيجن اللغة Delphi وسنستخدم المكتبة DES.pas (وهي مرفقة).
- يمكن كتابة كود توليد السريال كالتالي :

```
Uses
…., DES;
Function GetSerial ( Name: String ): String;
Var
Key,Temp, Serial :String;
I: Integer;
begin
key:='770video920avi532';
Temp:= EncryStr(Name ,key);
// Convert To Hex
For i:=1 to Length(Temp) do
begin
Serial:=Serial + IntToHex (ord(Temp[i]),2);
end;
// Return
Result:= Serial;
End;
```

معظم البرمجيات التي تستخدم خوارزميات تشفير في توليد السريال, تحتاج فقط إلى فهم أساسيات الخوارزمية وثوابتها ومتغيراتها. وكتابة كيجن لها أمر بمنتهى السهولة، مع توفر المكتبة الخاصة بلغتك في البرمجة.

كسر الكتب الإلكترونية المصنوعة ببرنامج eBook Workshop

كالعادة نحاول تبسيط الأمور لتظهر بصورة أوضح؛ لذلك سنحاول في هذا الدرس أن نطرح حلول لكل التساؤلات التي قد تأتي في ذهنك, لذلك نطلب منك القليل من التركيز.

مراحل الدرس :

- OllyDbg any version.
- MASM + RadAsm.
- AT4RE FastScanner v 2.0 .

(التحميل من مرفقات الكتاب).

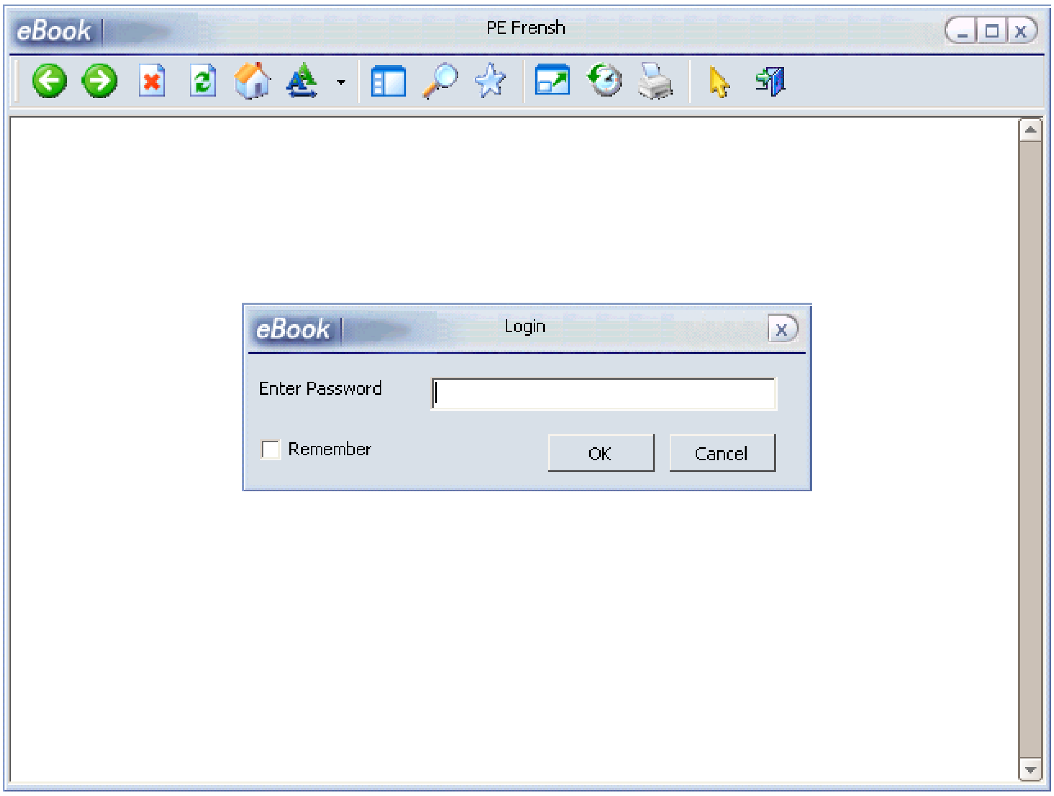
I. تعريف الكتب الإلكترونية

الكتب الإلكترونية مثلها مثل الكتب الورقية، لكنها بصفة حديثة رقمية، تترجم بياناتها كما يترجم أي برنامج، وهي عدة أشكال ولكلٍ خوارزميته، ونحن على يقين بانك رأيت أكثر من ثلاثة أصناف على الأقل من الكتب على هيئات مختلفة، مثلا : DOC ، PDF ، EXE كما رأيت أيضاً برامج للتحويل من هيئات الى أخرى.

موضوع اليوم يتحدث عن الكتب من هيئة ملف تنفيذي يحتوي صفحات الكتب بشكل منظم وسهل الاستعمال. هناك عدة برامج تنتج هذا الصنف من الكتب، وأشهرها Ebook Workshop الذي سنعتمده في الدرس.

لا يفترض بنا أن نشرح برنامج eBook Workshop لأنه ليس موضوعنا. لقد قمنا بصنع مثالين سنستغلهم في الدرس.

سنلقي نظرة على المثال الأول, وستجد مجلد باسم Ebook To Crack مرفق مع الدرس قم باختيار Exemple 1.exe وهو أبسط مثال بكلمة سر :



كما ترى أول ما يطالب به هو التسجيل ويطلب فقط كلمة السر. منطقيا وكما جرت العادة سنبحث عن شيء يأخذنا لهذه النافذة وهي Password أليس كذلك؟!

2. فك الكتب يدوياً في الذاكرة وتحليله

كما ترى فقد ذكرنا كلمة فك ولم نشر الى نوع التشفير او الضغط، وسنترك الأمر لـــ AT4RE FastScanner الإصدار الثاني.

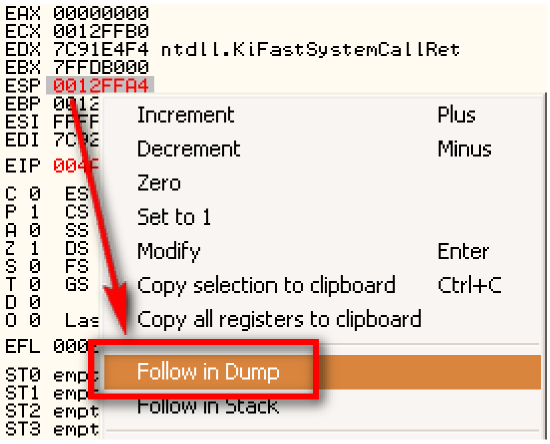
قم بفحصه لنعرف الضغط او التشفير :



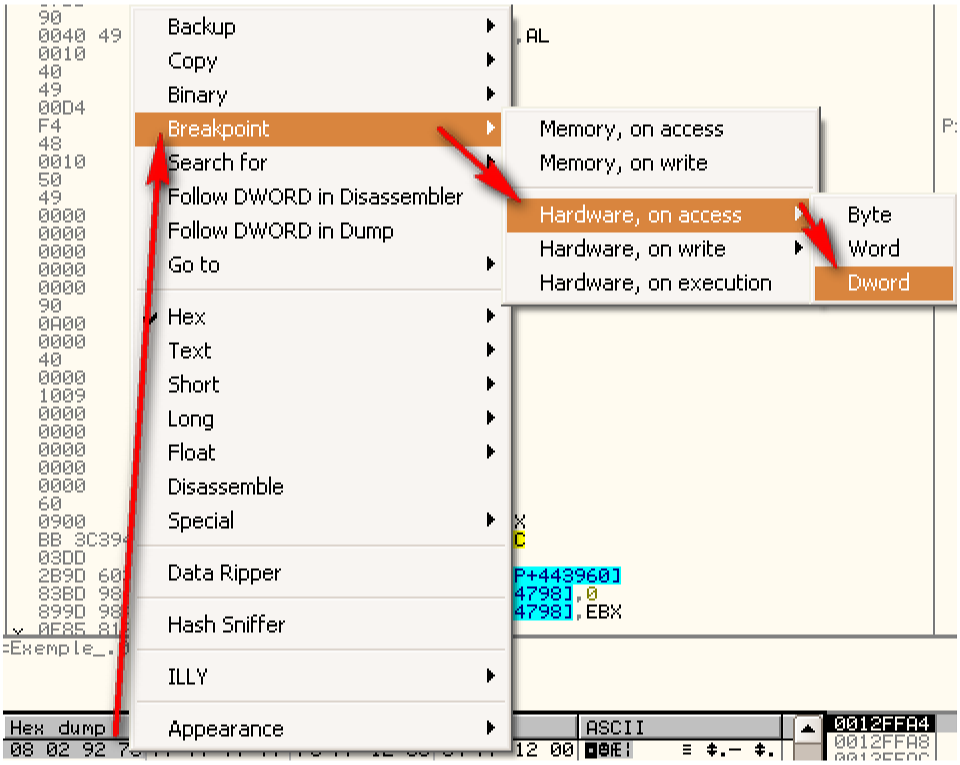
هذا الكتاب مضغوط بـــ ASPack وهو ضاغط مشهور كما تعلم ودروس فكه كثيرة جداً، لكن هناك سر في فك هذه الأمثلة سرنشير إليها في محلها بإذن الله.

بعد أن عرفنا نوع الضغط وطريقة الفك البسيطة المشروحة في النسخة الأولى من الكتاب، يبقى علينا التطبيق الحرفي للطريقة لنحصل على البرنامج الأصلي (مفكوك في الذاكرة).

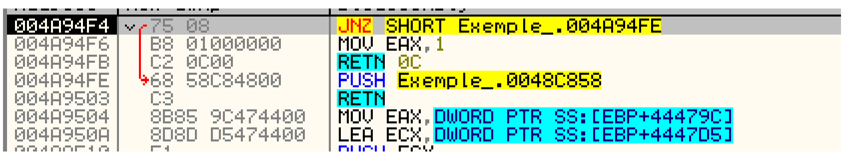
حمل الكتاب الأول إلى برنامج التنقيح ثم F8 ثم :



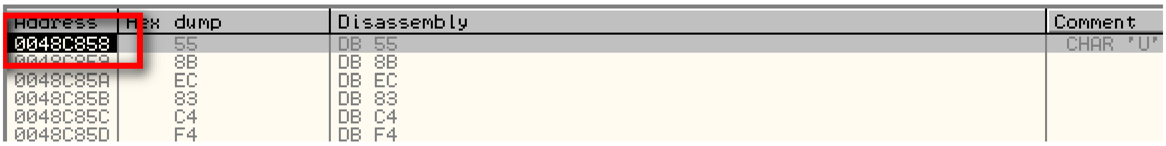
ثم في قسم الـــ Dump :



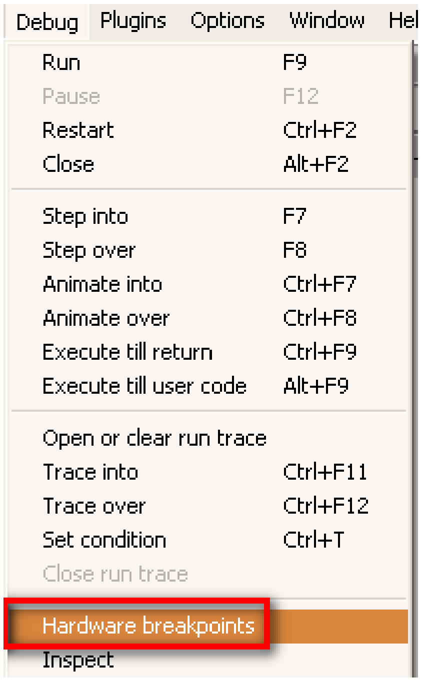
لاحظ لقد ضللنا البتات الأربعة الأولى. الآن F9 :



ثم F8 ثلاث مرات، وها أنت في OEP :



قم بعمل تحليل للكود، وذلك بالضغط على Ctrl + A ثم أزل نقطة التوقف من نوع HardWare التي وضعناها، كما يلي : اتجه لقائمة Debug :



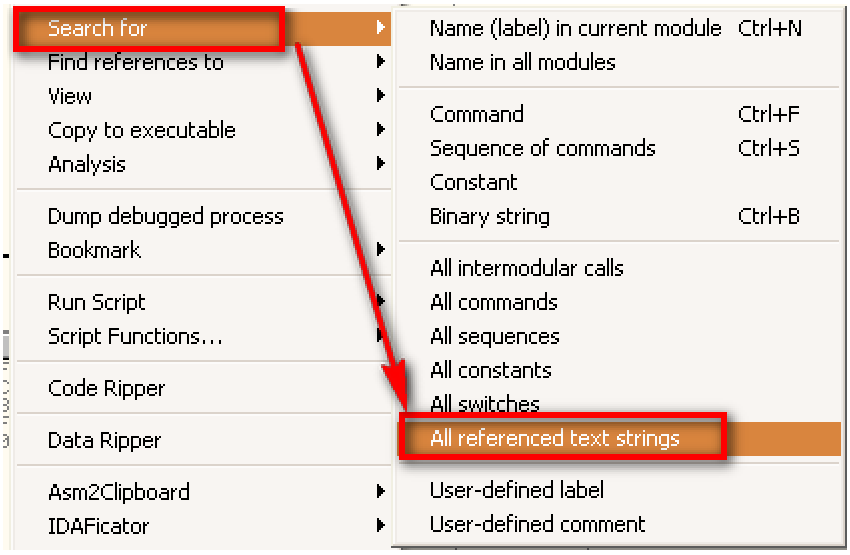
ثم :



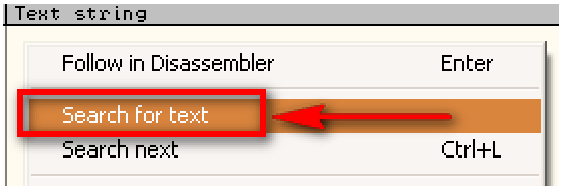
والآن لدينا الكتاب جاهز للتنقيح والكسر.

سنقوم الان بالبحث عن الكلمة المفتاح لكسر الكتاب وهي Password ضمن نصوص البرنامج :

قم بالضغط على الزر الأيمن ضمن قسم الكود :

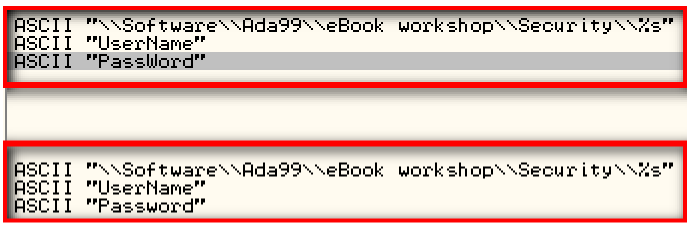


ستخرج لك نافذة النصوص اصعد للأعلى وابدأ البحث المتتالي عن كلمة Password :





وبالبحث المتتالي نجد شيئا مثيراً وواضحاً :



الآن تعرف ما عليك فعله, وهو وضع نقط توقف في كلا المؤشرين لكلمة Password ثم الضغط على F9 , ماذا حصل؟ لقد توقف في إحداهما. لاحظ معنا في الأول لما فتحنا الكتاب بدون المنقح خرجت لنا نافذة التسجيل، أليس كذلك؟ وها نحن متوقفون

الآن في المنقح – في شيء مهم لكسر الكتاب– و ما علينا سوى جعل المنقح يتابع تنقيح الكتاب تلقائياً لنرى موقع خروج النافذة المزعجة.

اضغط Ctrl + F8 ثم لاحظ مكان التوقف جيدا :

```
0048AE11  .  E8 1292F7FF    CALL Exemple_.00404028
0048AE16  .v 75 52          JNZ SHORT Exemple_.0048AE6A
0048AE18  .  B3 01          MOV BL,1
0048AE1A  .v EB 4E          JMP SHORT Exemple_.0048AE6A
0048AE1C  >  8D55 EC        LEA EDX,[LOCAL.5]
0048AE1F  .  A1 E8E44800    MOV EAX,DWORD PTR DS:[48E4E8]
0048AE24  .  8B00           MOV EAX,DWORD PTR DS:[EAX]
0048AE26  .  8B80 18030000  MOV EAX,DWORD PTR DS:[EAX+318]
0048AE2C  .  E8 67F0F9FF    CALL Exemple_.00429E98
0048AE31  .  8B45 EC        MOV EAX,[LOCAL.5]
0048AE34  .  8D4D F0        LEA ECX,[LOCAL.4]
0048AE37  .  8B15 54E54800  MOV EDX,DWORD PTR DS:[48E554]
0048AE3D  .  8B12           MOV EDX,DWORD PTR DS:[EDX]
0048AE3F  .  E8 D8080000    CALL Exemple_.0048B71C
0048AE44  .  8B45 F0        MOV EAX,[LOCAL.4]
0048AE47  .  50             PUSH EAX
0048AE48  .  8D55 E8        LEA EDX,[LOCAL.6]
0048AE4B  .  A1 E8E44800    MOV EAX,DWORD PTR DS:[48E4E8]
0048AE50  .  8B00           MOV EAX,DWORD PTR DS:[EAX]
0048AE52  .  8B80 1C030000  MOV EAX,DWORD PTR DS:[EAX+31C]
0048AE58  .  E8 3BF0F9FF    CALL Exemple_.00429E98
0048AE5D  .  8B55 E8        MOV EDX,[LOCAL.6]
0048AE60  .  58             POP EAX
0048AE61  .  E8 C291F7FF    CALL Exemple_.00404028
0048AE66  .v 75 02          JNZ SHORT Exemple_.0048AE6A
0048AE68  .  B3 01          MOV BL,1
0048AE6A  >  84DB           TEST BL,BL
0048AE6C  .v 0F85 FE000000  JNZ Exemple_.0048AF70
0048AE72  .  A1 E8E44800    MOV EAX,DWORD PTR DS:[48E4E8]
0048AE77  .  8B00           MOV EAX,DWORD PTR DS:[EAX]
0048AE79  .  8B10           MOV EDX,DWORD PTR DS:[EAX]
0048AE7B  .  FF92 D8000000  CALL DWORD PTR DS:[EDX+D8]
0048AE81  .  48             DEC EAX
0048AE82  .v 0F85 E8000000  JNZ Exemple_.0048AF70
0048AE88  .  A1 84F94800    MOV EAX,DWORD PTR DS:[48F984]
```

الكتاب أطلق نافذته المزعجة للتسجيل، وكما ترى فالمنقح توقف عند استدعائها، وستجد قبلها قفزات مشروطة، والقفزة الواضحة التي لا تجعل الكتاب مسجلاً هي التي لا تدع المسجل BL يأخذ قيمة 1 (صحيح) :

JNZ SHORT 0048AE6A

ما علينا سوى عكسها او جعلها لا تقفز دائماً، لنعكسها, لماذا نعكسها بدل أن نجعلها لا تقفز دائماً؟ الجواب: هو عدد البايتات التي سنغيرها؛ ففي حال عكسها فقط سنحصل على تعديل بايت واحد، أما في حال جعلها لا تقفز ابدا فإننا سنغير على الأقل بايتين او ثلاثة.

لنعكسها الى :

JZ SHORT 0048AE6A

تجدها تغيرت الى :

```
0048AE0F  .  8B00           MOV EAX,DWORD PTR DS:[EAX]
0048AE11  .  E8 1292F7FF    CALL Exemple_.00404028
0048AE16   v 74 52          JE SHORT Exemple_.0048AE6A
0048AE18  .  B3 01          MOV BL,1
0048AE1A  .v EB 4E          JMP SHORT Exemple_.0048AE6A
0048AE1C  >  8D55 EC        LEA EDX,[LOCAL.5]
```

كل ما عليك تذكره هو :

	العنوان 0048AE16h والبايت الذي غيرنا من 75h الى 74h.

اضغط F9 لنجرب الكتاب :

ماذا ترى؟ الكتاب مكسور ويمكنك تصفحه، رائع جداً !!

بهذا ننهي هذه الفقرة ونمر للتالية.

3. كتابة لودر للكتاب

أتتذكر ما قلناه في البداية :

" هذا الكتاب مضغوط بـــ ASPack وهو ضاغط مشهور كما تعلم ودروس فكه كثيرة جداً، لكن هناك سر في فك هذه الأمثلة سنشير إليها في محلها بإذن الله. "

هنا سنجيب : ببساطة فإن هذا النوع من الكتب لا يستخدم ضغط فقط، بل يعتمد تقنية الـــ Overlay وهي تقنية تتطلب بعض المهارات في التعامل معها، كما أن هناك درس للأخ Atlasse ، وشرح من الأستاذ dj-siba حول هذه التقنية.

بذلك نستنتج أنه ليس من السهل فكه يدوياً، لكن ما نتفق عليه أننا كسرناه في الذاكرة ولدينا بديل للـــ Patch وهنا تتجلى قيمته، وهو الـــ Loader وهذا البريمج ببساطة يقوم بالتعديل على برنامج ما في الذاكرة مباشرة.

كما تلاحظ من عنوان الفقرة " كتابة لودر للكتب " وليس صنعه ببرنامج ما.

الآن سنقوم ببرمجة Loader للكتاب الأول بالأسمبلي، وباستخدام دوال قد تم شرحها في الإصدار الأول من الكتاب في فقرة اللودر؛ لذلك سنضع الكود مباشرة، وسنضع ملاحظات فقط :

```
.386
.model flat,stdcall
option casemap:none
include         windows.inc
include         kernel32.inc
include         user32.inc
includelib      kernel32.lib
includelib      user32.lib

.data?
sinfo       STARTUPINFO     <>
pinfo       PROCESS_INFORMATION     <>
BytesWritten        dd          ?
.data
TargetBook      db          "Exemple 1.exe",0
PatchByte       db          74h
AdressVA        dd          0048AE16h

.code
start:
invoke CreateProcess,addr TargetBook,0,0,0,0,0,0,0,addr sinfo,addr pinfo
invoke Sleep,100
invoke SuspendThread,pinfo.hThread
invoke WriteProcessMemory,pinfo.hProcess,AdressVA,addr PatchByte,1,0
invoke ResumeThread,pinfo.hThread
invoke ExitProcess,NULL
end start
```

ما قد تستغرب منه هو الدالة Sleep ولماذا استعمإنها ؟

كما نعرف عن البرامج المضغوطة والمشفرة فإنها تأخذ وقتً لتفك في الذاكرة, لذلك سنقوم بإيقاف اللودر مدة زمنية معينة قصيرة جداً ثم نكمل عملنا بسلام .

الآن جرب اللودر : ضعه بجانب الكتاب ثم افتحه!! مبروك عليك تم كسره و لم يعد يظهر لك النافذة المزعجة.

9. استخراج السريال

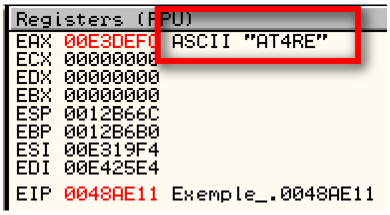
الأمر بسيط جدا هذه المرة، في المرة السابقة، أي في الفقرة السابقة، جعلنا المنقح يدلنا على مكان استدعاء نافذة التسجيل، و لم نشاهد المقارنات التي يقوم بها، أما في هذه الفقرة فسنقوم بالتتبع بأنفسنا سطراً سطراً، وسنرى ماذا سيحصل ؟

أعد تشغيل المثال، وقم بإعادة نفس المراحل في الفقرة السابقة، ولكن لا تضغط هذه المرة Ctrl + F8 بل قم بالتتبع بـــ F8 :

قبل القفزة التي عدلنا :

```
0048AE07  .   8B55 F4        MOV EDX,[LOCAL.3]
0048AE0A  .   A1 54E54800    MOV EAX,DWORD PTR DS:[48E554]
0048AE11  .   E8 1292F7FF    CALL Exemple_.00404028
0048AE16  .v  75 52          JNZ SHORT Exemple_.0048AE6A
0048AE18  .   B3 01          MOV BL,1
0048AE1A  .v  EB 4E          JMP SHORT Exemple_.0048AE6A
0048AE1F  .   A1 E8E44800    MOV EAX,DWORD PTR DS:[48E4E8]
```

انظر إلى المسجل EAX :



ستجد القارنة، والسريال في المسجل. ها قد وجدنا السريال بسهولة، وهو AT4RE جربه، ومبلوك عليك.

5. كتابة Sniffer للسريال

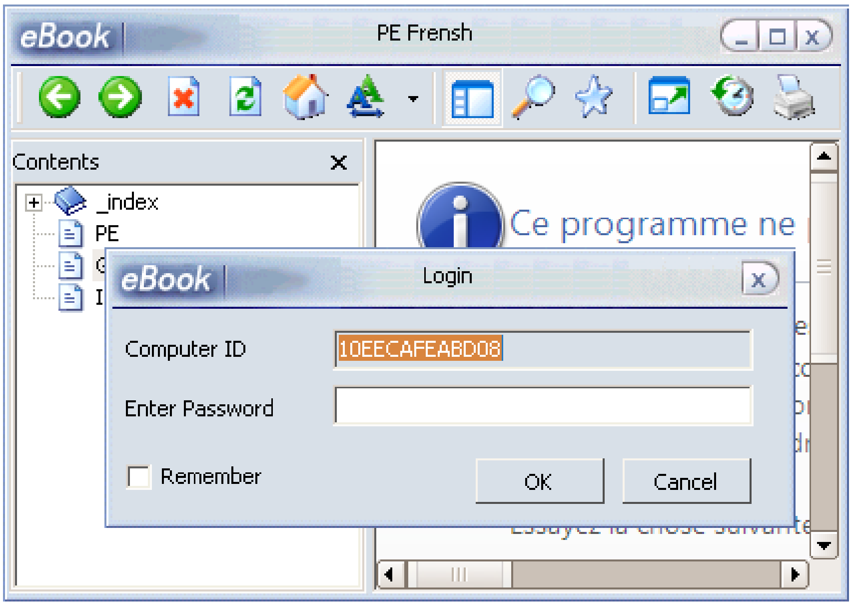
سيضع الكود بدون شرح؛ لأنه قد تم شرح هذه الفقرة في الإصدار الأول من الكتاب :

```
.386
.model flat,stdcall
option casemap:none
include         windows.inc
include         kernel32.inc
include         user32.inc
includelib      kernel32.lib
includelib      user32.lib
.data?
sinfo       STARTUPINFO     <>
pinfo       PROCESS_INFORMATION     <>
cntx            CONTEXT <>
.data
TargetBook      db          "Exemple 1.exe",0
PatchByte       db          74h
AdressVA        dd          0048AE11h
Serial db 512 dup(?)
.const
InfiniteLoop        db                      0EBh,0FEh
.code
start:
invoke CreateProcess,addr TargetBook,0,0,0,0,0,0,0,addr sinfo,addr pinfo
invoke Sleep,100
invoke WriteProcessMemory,pinfo.hProcess,AdressVA,addr InfiniteLoop,2,0
invoke ResumeThread,pinfo.hThread
mov cntx.ContextFlags,CONTEXT_FULL
GetContextLoop:
invoke GetThreadContext,pinfo.hThread,addr cntx
mov esi,AdressVA
cmp cntx.regEip,esi
je SetSerial
jmp GetContextLoop
SetSerial:
invoke SuspendThread,pinfo.hThread
invoke ReadProcessMemory,pinfo.hProcess,cntx.regEax,addr Serial,999,0
invoke MessageBox,0,addr Serial,0,MB_OK
invoke ExitProcess,0
end start
```

جرب الآن Sniffer وسيخرج لك سريال للكتاب، وبهذا ننهي فقرة Sniffer .

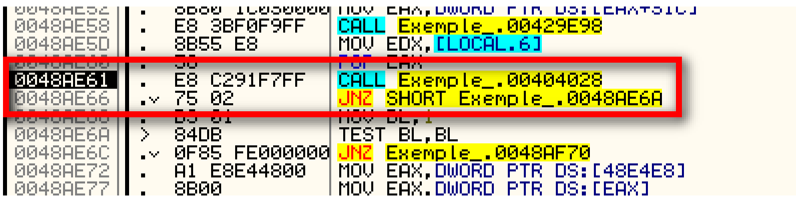
6. صنع الكيجن للكلب

لنفتح المثال الثاني سنجد أنه يطلب سريال متعلق بقيمة بدورها متعلقة باسم الجهاز :



ما علينا سوى جلب السريال أولاً، ثم سنفكر بالكيجن. كيف سنجلب السريال لهذا المثال؟ الجواب بسيط جداً: بنفس طريقة المثال الأول سنجد السريال للمثال الثاني.

وسنجده عند :



في المسجل EAX لكنه يتغير من جهاز لآخر؛ فمثلاً على جهازي أجد :

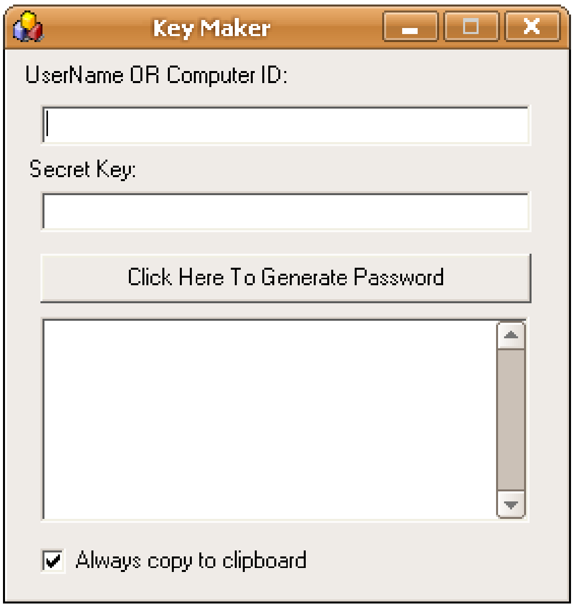
B087B3739356A0

أما لديك فلن يشتغل؛ لذلك يلزمنا كيجن. قبل البدء في مراحل الكيجن لنقم بصنع Sniffer للكتاب الثاني بنفس الطريقة تماماً، ويكفي تغيير العنوان AdressVA الى 0048AE61h . أما اللودر الأول الذي برمجناه فيصلح له؛ لأن عنوان المقارنة لا يتغير. أما في حال لو تغير فيكفي تغييره في السورس كود ثم جمع اللودر.

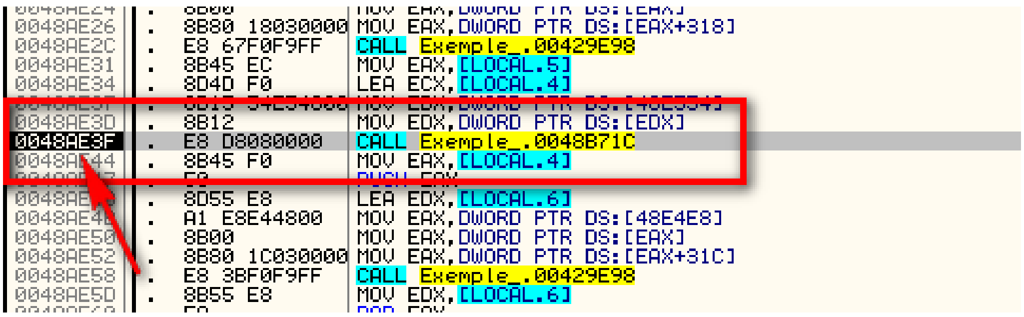
لنمر الى صنع الكيجن – وليس برمجته – يلزمك بريمجين يأتيان مع برنامج Ebook WorkShop وهما :

Key Maker و GetID

تجدهما مرفقين مع الدرس. حسناً لنفتح Key Maker :



غريب لا يطلب اسم الجهاز فقط، بل يطلب المفتاح أيضاً لتوليد السريال!! وهذا المفتاح يكون ثابتاً في البرنامج، وستجده بالتنقيح بسهولة عند العنوان :

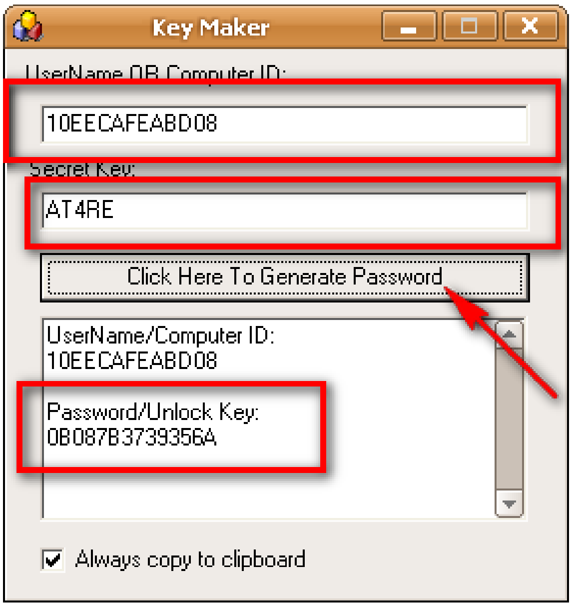


في المسجل EAX . في مثالنا المفتاح هو AT4RE .

والآن بعدما وجدنا المفتاح يلزمنا الرقم التسلسلي للجهاز سنحصل عليه بـــ GetID:



اضغط على OK لنسخه، ثم اتجه لـــ Key Maker لنولد السريال :



وأخيراً هاهو السريال الصحيح لجهازي بطريقة بسيطة جداً. جرب وستحصل على سريال خاص بجهازك.

الباب الثامن: تصميم سكينات للباتشات

تصميم سكين لبرنامج dUP باستخدام الفوتوشوب

سنقدم هنا شرحاً للمراحل العملية لتصميم سكين لبرنامج dup والذي يستخدم لصناعة باتشات ولودرات.

- برنامج dUP Version 2.19.
- برنامج Adobe Photoshop CS3.
- برنامج Resource Hacker.
- برنامج RGNCreator.

(التحميل من مرفقات الكتاب).

سيتناول هذا الدرس النقاط الرئيسية التالية :

1) ملاحظات حول التصميم.
2) تهيئة السكين.
3) تركيب السكين.

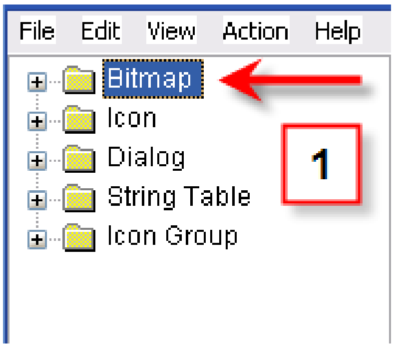
I. أولاً : ملاحظات حول التصميم

– طبعاً تختلف رؤية التصميم ومستوى التقنية باختلاف الأشخاص والأذواق، وسيقتصر حديثنا على الجوانب المتعلقة بتركيب السكين على برنامج dUP .

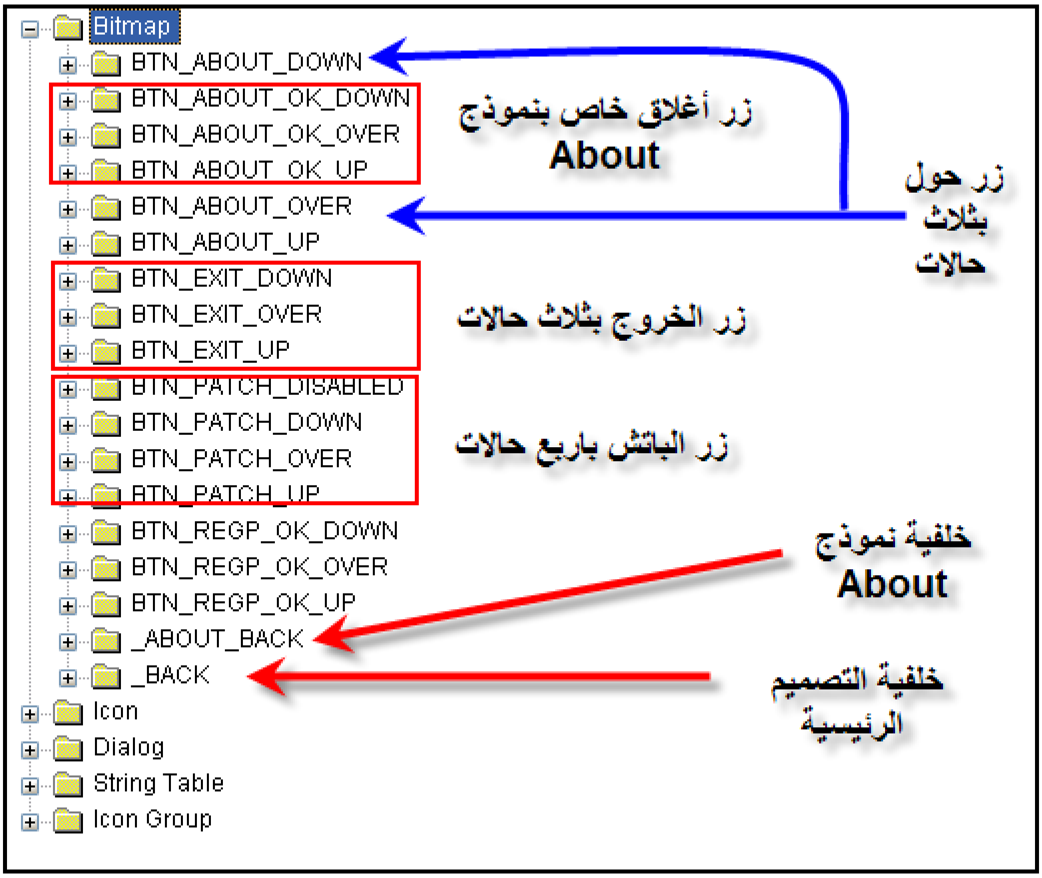
– أرفقنا مع الدرس تصميماً مع الملف المصدري بلاحقة PSD ليتم العمل عليه أثناء الشرح.

– سنعتمد على سكين موجود مسبقاً, يأتي مع برنامج dUP وهو VistaSkin.res , وسنحاول استبدال الصور الخاصة بتصميمنا بالصور التي تحتويه.

– نفتح الملف VistaSkin.res ببرنامج Resource Hacker ونشاهد ما هي الصور الواجب علينا تصميمها ؟



– نفتح المجلد Bitmap ونلقي نظرة على الصور الواجب تزويد البرنامج بها :



إذن نحن بحاجة إلى الصور التالية في النموذج الرئيسي :

1) خلفية للتصميم.

2) زر Patch بأربع حالات :

Up : الوضع الطبيعي للزر.

Down : ضغط الزر.

Over : المرور فوق الزر بالفأرة.

Disable : الزر غير مفعل (وهذه ينفرد بها زر Patch).

3) زر About بثلاث حالات : (Up – Down – Over).

4) زر EXIT بثلاث حالات : (Up – Down – Over).

وأيضاً الصور التالية في نموذج About :

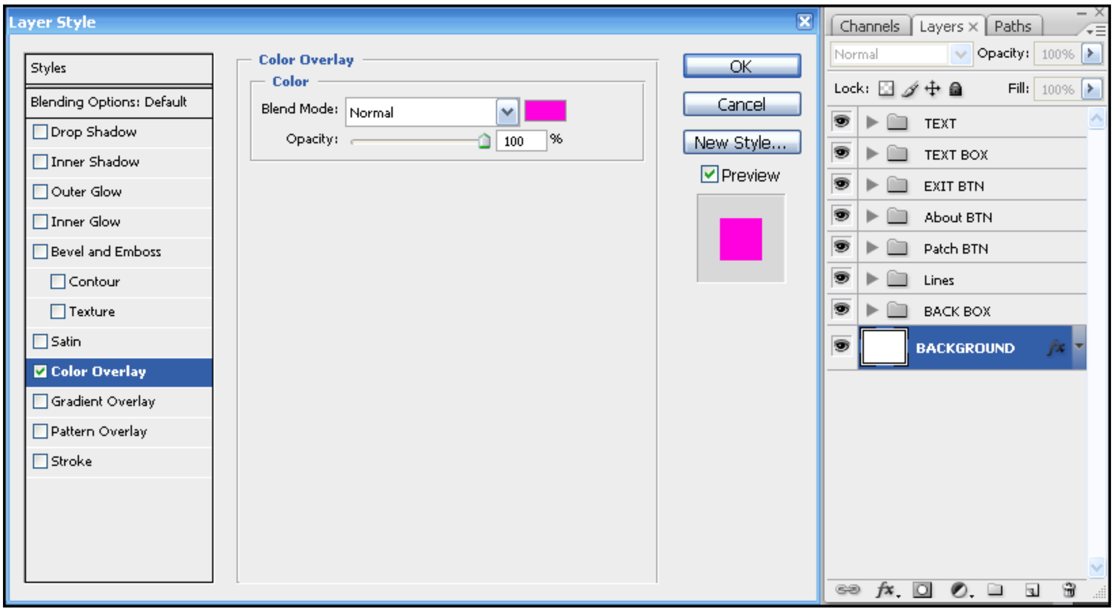
1) خلفية لنموذج About.

2) زر إغلاق لنموذج About بثلاث حالات أيضاً.

– نفتح ملف SourceSkin.Psd ببرنامج Photoshop ونبدأ العمل .

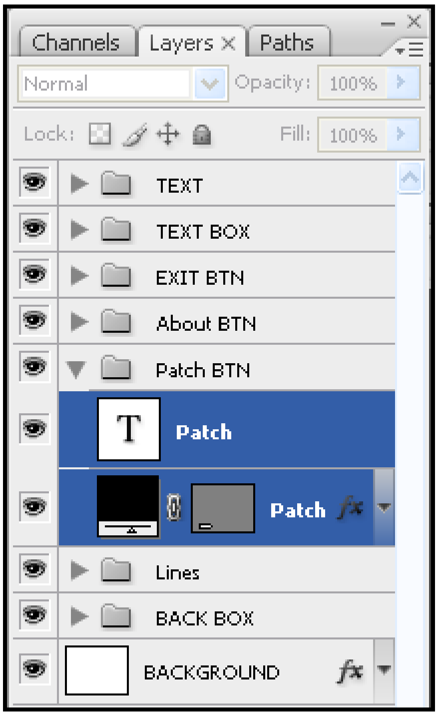
– يزودنا برنامج dup بميزة تفريغ لون معين من التصميم, من خلال إنشاء ملف بلاحقة RGN وهنا يجب الانتباه إلى أن يكون هذا اللون غير مستخدم في التصميم.

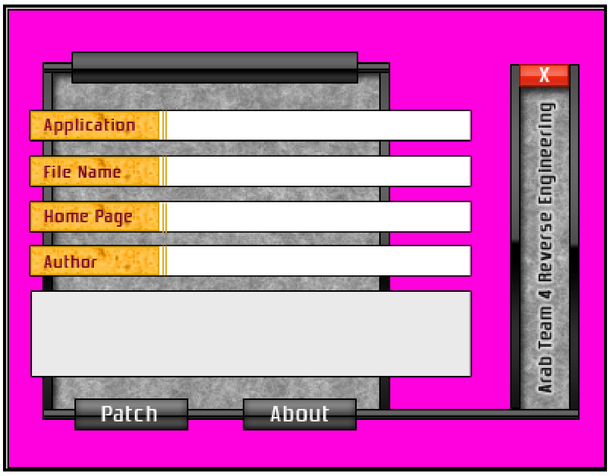
– حاول جعل لون الخلفية (الطبقة Background) بلون مشابه للصورة التالية من خلال تعيين الخاصية Color Overly من النافذة Layer Style والتي يمكن أن تظهرها بالنقر المزدوج على الطبقة :



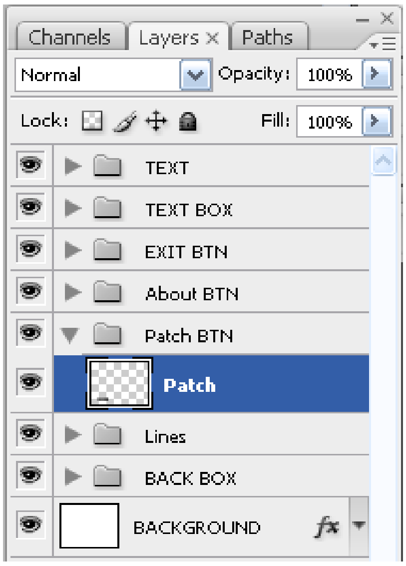
– الآن سنحاول إنشاء الحالات الأربع للزر Patch (ما يطبق عليه يمكن أن يطبق على بقية الأزرار)، وسنستخدم هنا طريقة بسيطة وسريعة.

– افتح المجلد PATCH BTN في نافذة Layers واختر النص Patch والشكل Patch بالاستعانة بالضغط على الزر Ctrl أثناء ضغط الفأرة.

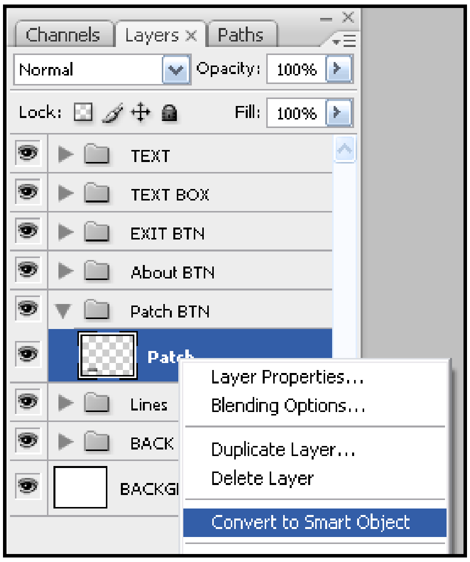




الآن اضغط على المفاتيح Ctrl + E لدمج الطبقتين مع بعضهما، وستشاهد النتيجة التالية :

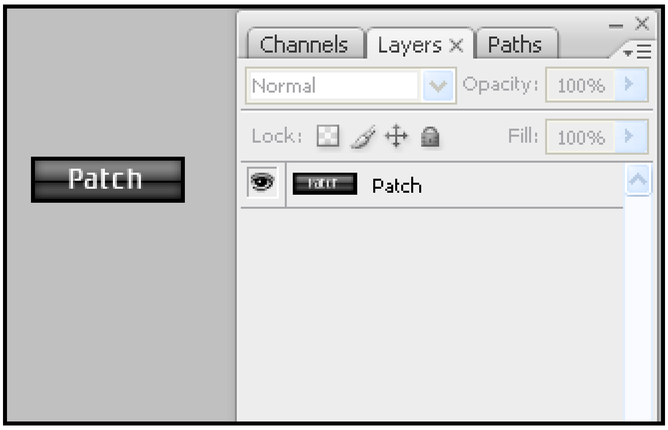


اضغط عليها بزر الفأرة الأيمن، واختر Convert To Smart Object :



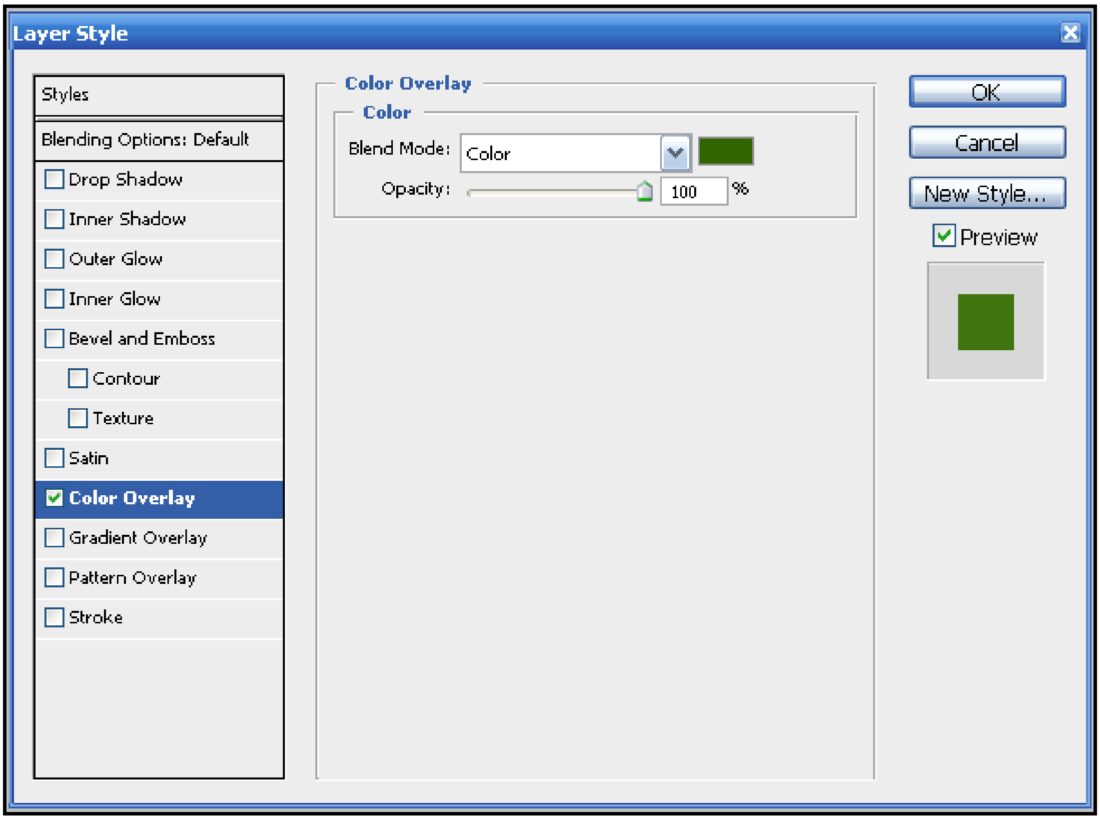
مرة أخرى hضغط بزر الفأرة الأيمن، واختر Edit Contents :

الآن الزر فُتح بمشروع جديد، وهو قابل للتحرير مفرداً :

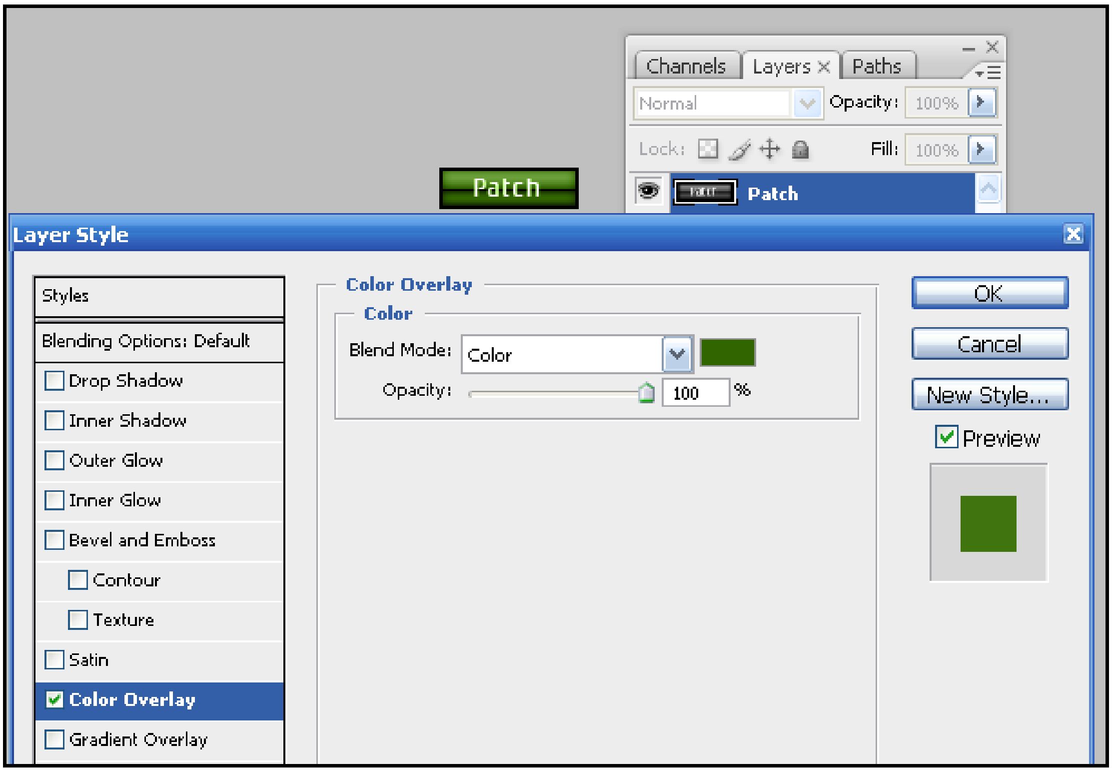


– احفظ الملف الآن بلاحقة BMP باستخدام : File > Save As أو استخدم تركيبة المفاتيح Ctrl + Shift + S واحفظها بالاسم **Patch_Up.Bmp** وهو الحالة الأولى للزر.

– انقر على الطبقة Patch نقراً مزدوجاً لتخرج لك النافذة LayerStyle اختر منها الباب Color Overlay واختر لوناً وليكن الأخضر :



اضغط على القائمة المنسدلة Blend Mode واختر الخاصية Color :

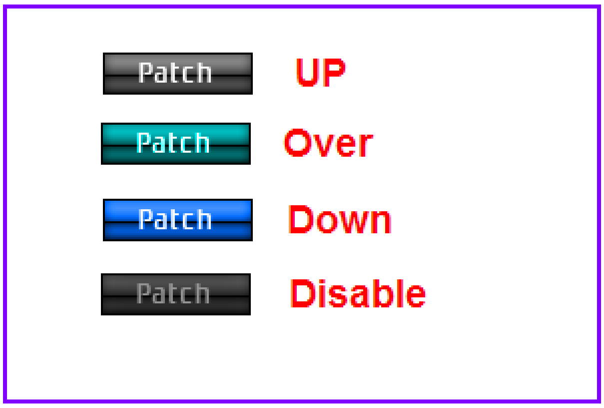


– احفظ الملف الجديد بلاحقة Bmp وباسم : Patch_Over .

– كرر العملية السابقة من أجل الزر Patch_Down مع اختيار لون آخر أو تأثير مختلف.

– بالنسبة للحالة Patch_Disable يمكن إضافة التأثير Satin مع اختيار اللون الرمادي والخاصية Distance = 0 .

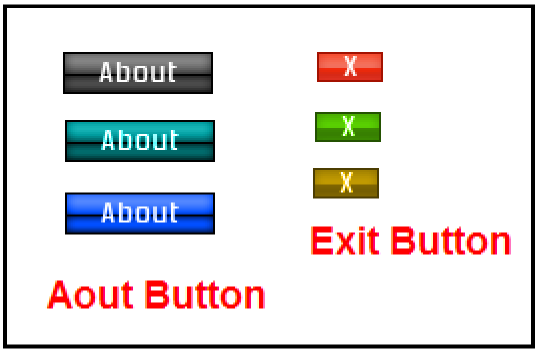
– سيكون الشكل النهائي للحالات الأربع للزر مشابه للصورة التالية :



– يجب الانتباه إلى حفظ الألوان المطبقة على الزر Patch لتطبقها على الزر About فشيء جميل أن يحمل الاثنين نفس اللون.

– كما نود الإشارة إلى إمكانية تحويل المستطيل إلى Smart Object ثم إضافة نص عليه، وهذا يكون أكثر فعالية، ويسمح لك بإضافة تأثير منفصل على النص، ولكننا اتبعنا هذه الطريقة حتى لا يطول الشرح كثيراً.

– الآن يمكنك تكرار نفس الخطوات من أجل الأزرار About و Exit :

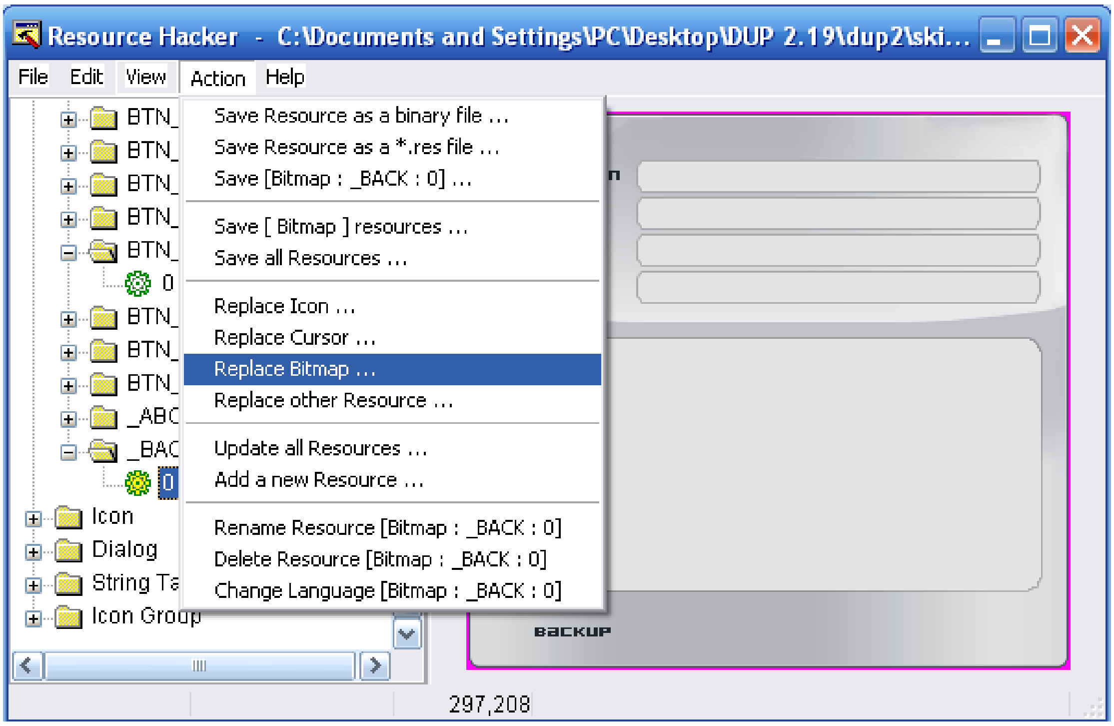


– أما نموذج About فيتبع نفس التصميم المستخدم في النموذج الرئيسي، مع الانتباه إلى عدم إمكانية استخدام خاصية أخذ شكل معين (RGN) ولن نتطرق لشرحه هنا.

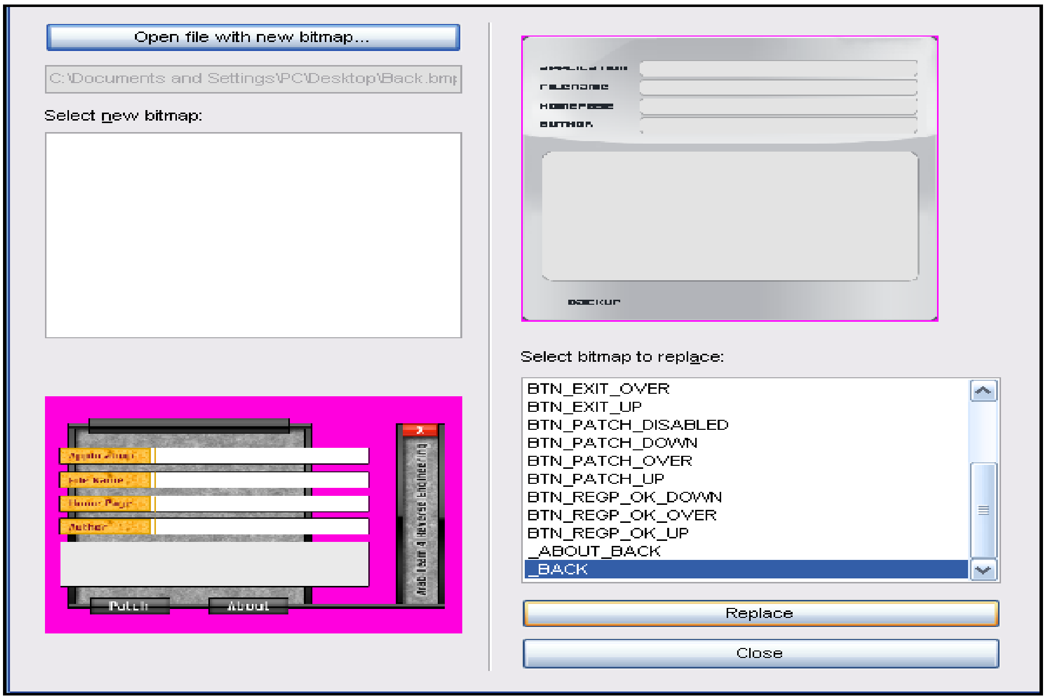
2. ثانياً : تهيئة السكين

سنقوم بفتح السكين VistaSkin.res ببرنامج Resource Hacker ونقوم باستبدال الصور التي صممناها بالصور الموجودة، وفق الخطوات التالية :

– نعين الصورة Back_ ثم نختر من القائمة Action > Replace Bitmap :

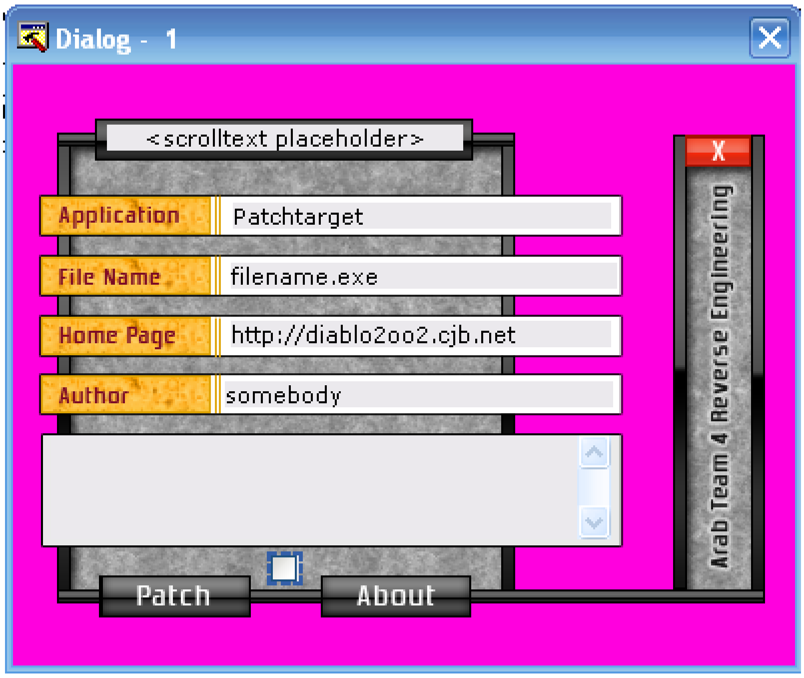


– بعد ذلك نختار تصميم الخلفية الخاص بنا ونضغط Replace :



– نكرر الخطوات السابقة مع كل الصور التي تمثل حالات الأزرار، والتي قمنا بتصميمها منذ قليل، مع الانتباه إلى التسميات الموافقة لكل زر وحالته.

– بعد الانتهاء ننتقل إلى تعديل النماذج Dialog لوضع العناصر في أماكنها المطابقة لتصميمنا، سنحاول جعل النموذج والعناصر مشابهة للشكل التالي :



– نضغط على زر Compile Script .

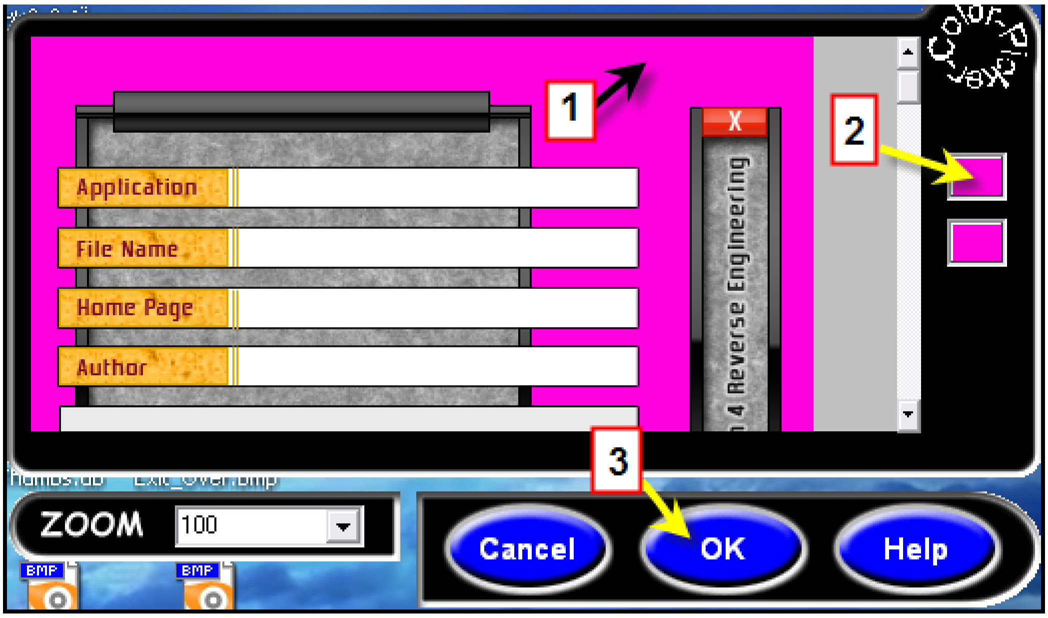
– الآن نحفظ السكين باسم جديد، وليكن MySkin.res من خلال الأمر : File > Save As .

– وبهذا نكون قد انتهينا من تهيئة السكين.

– سننتقل الآن إلى عمل ملف RGN والذي يستخدم لتفريغ الشكل من اللون البنفسجي.

سنستخدم لذلك البرنامج RGNCreator :

– نفتح صورة الخلفية من خلال Browse ثم نضغط الزر Pick ونختار اللون البنفسجي باستخدام شافطة الألوان ثم نضغط زر الفأرة الأيمن ونختر Ok :



بعدها نقوم بحفظها بالضغط على زر Create باسم My_Rgn.RGN .

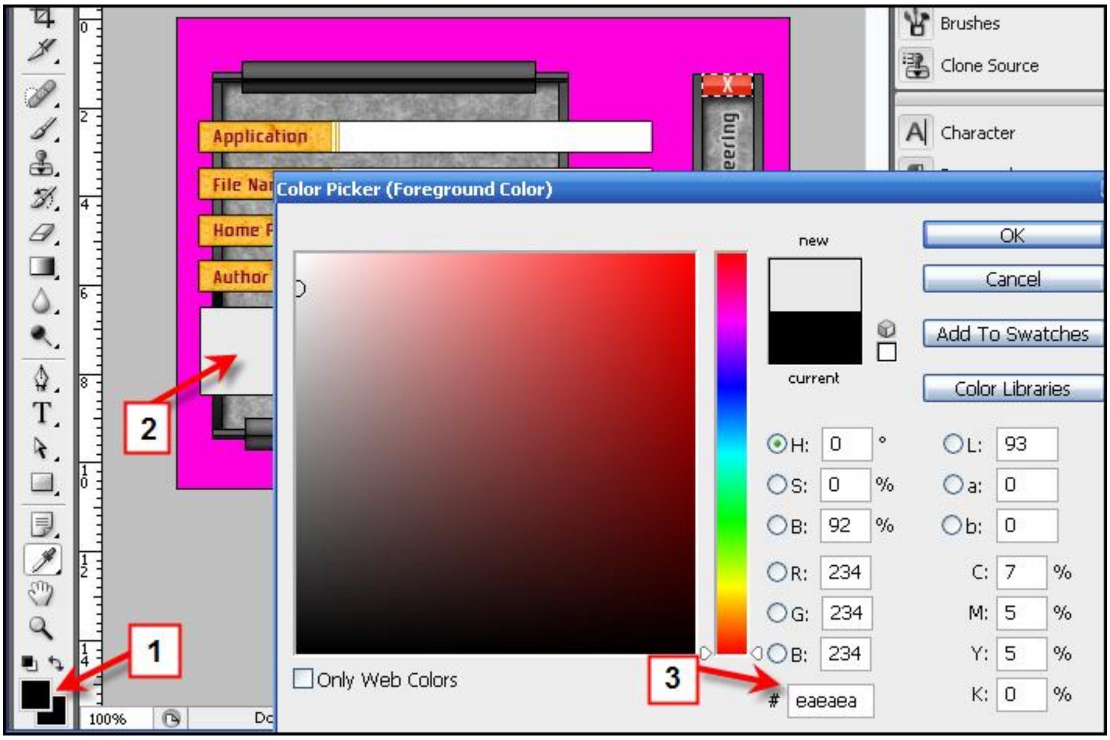
3. ثالثاً : تركيب السكين

- نتوجه إلى برنامج Dup وندخل معلومات البرنامج من خلال الزر New Project .
- ننتقل بعدها إلى لوحة Settings ونزود البرنامج بملف السكين الذي أنشأناه MySkin.res وبملف RGN (MyRgn.RGN).
- أما بالنسبة لألوان المربعات النصية ولون الكتابة، فيجب ضبطها لتتوافق مع التصميم، وللحصول على اللون بدقة نتبع ما يلي ضمن الفوتوشوب :

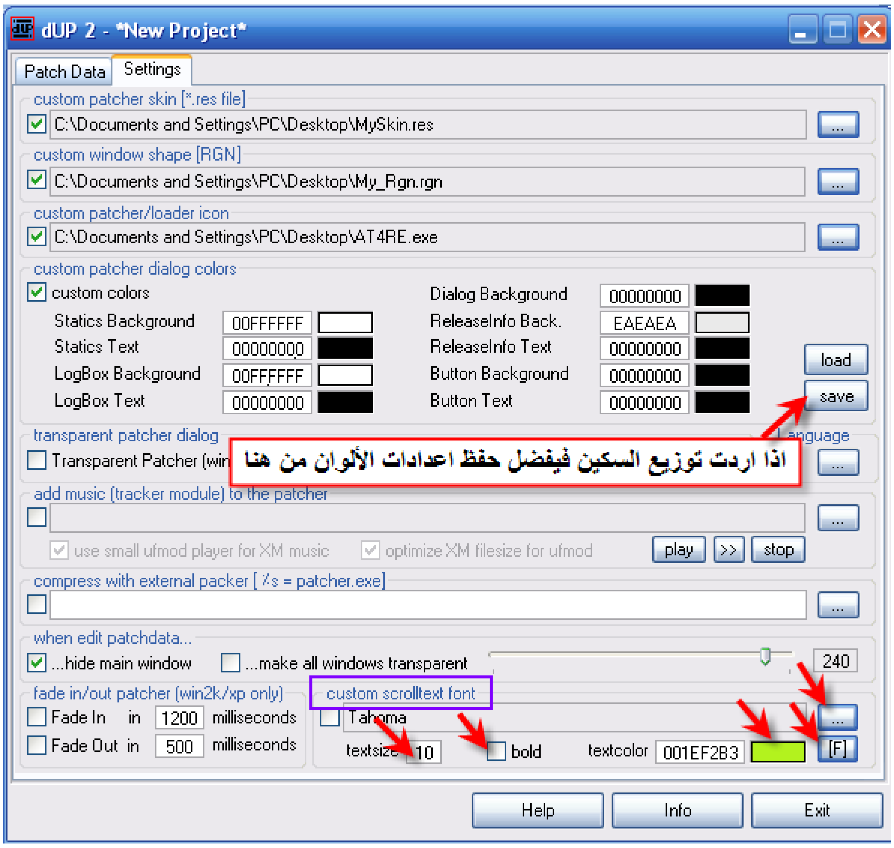
1 - نضغط على المربعين في الزاوية اليسرى.

2 - نختار اللون بشافطة الألوان .

3 - ننسخ اللون من المربع السفلي (الستة عشري) ونلصقه في Dup .



– يمكننا أيضاً تغير لون وحجم الخط ونوعه للنص المتحرك Scroll text حسب ذوقنا ورؤيتنا الفنية، وسنحاول ضبط الإعدادات كالتالي :



- الآن الملف جاهز للانطلاق من خلال الضغط على الزر Create Patch وهذه صورة للسكين بعد التركيب :



تصميم سكين لبرنامج Uppp باستخدام الفوتوشوب

لمن لا يعرف برنامج Uppp فهو برنامج صانع للباش يدعم عدة أنواع Offset patch وS+R Patch و Registry patch و FileDrop ويدعم صور الـــ PNG فقط في السكينات وهو من برمجة UFO-Pu55y من فريق SND .

الأدوات والملفات المرفقة :



- uPPP SkinHelper v0.1.
- الملفات الخاصة بالسكين.
- ملف التصميم الخاص بالفوتوشوب (PSD.).



(التحميل من مرفقات الكتاب).

I. خطوات العمل

1. قم بتصميم سكين بأي برنامج تصميم تجيد استخدامه (يمكنك الاستفادة من ملف الـــ PSD المرفق).
2. عمل السكين يتطلب حفظ 6 صور من نوع PNG بالشكل التالي :

1 - الخلفية.
2 - زر الباش.
3 - زر الخروج.
4 - زر عمل النسخة الاحتياطية (هذه الخاصية اختيارية يمكن الاستغناء عنها).
5 - زر إيقاف وتشغيل النشيد (هذه الخاصية اختيارية يمكن الاستغناء عنها).
6 - صورة صغيرة لمشاهدة السكين عند اختياريه في البرنامج (عادة تكون أبعادها 150 X 275).

ملاحظات خاصة بالصور :

يجب أن تكون جميع الصور من نوع 32 بت PNG ما عدا صورة مشاهدة السكين لا يشترط بها ذلك.

تحتوي الخلفية كما في المثال التالي على فراغ لاسم البرنامج ورابط الموقع والمؤلف والتاريخ (علماً بأن كل هذه الأمور اختيارية ويمكن الاستغناء عنها).

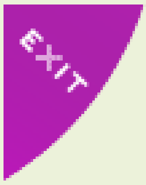


أما بالنسبة لزر الباش فيجب أن يكون كما في المثال التالي، يحتوي على الحركات الثلاث بالترتيب التالي (للأعلى ، مؤشر عليه ، للأسفل) ويجب أن يكون عرض الصورة (in pixels) عدداً زوجياً.

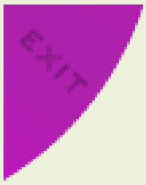


أما بالنسبة لزر الخروج فيجب أن يكون كما في المثال التالي، يحتوي على الحركات الثلاث بالترتيب

التالي (للأعلى ، مؤشر عليه ، للأسفل) ويجب أن يكون عرض الصورة (in pixels) عدداً زوجياً.







أما بالنسبة لزر النسخة الاحتياطية فيجب أن يكون كما في الصورة التالية، تكون الحالة العلوية هي : لا أريد أن ينتج نسخة احتياطية، والسفلية : أريد أن أنتج، ومرة أخرى يجب أن يكون العرض (in pixels) عددا زوجياً.



(قمنا بوضع اللون الرمادي كخلفية، لكي تظهر كل الأزرار في الشرح)

أما بالنسبة لزر الصوت، فهو شبيه بزر النسخة الاحتياطية، ولكن تغير الكتابة كما في الصورة التالية :



(و هنا أيضاً قمنا بوضع اللون الرمادي كخلفية، لكي تظهر كل الأزرار في الشرح)

صورة مشاهدة السكين كما قلنا صغيرة الأبعاد (150 X 275) فهي غير مهمة للسكين، ولكنها مهمة للبرنامج، فلا يمكن استدعاء السكين من دونها.



3. تحتاج أيضاً إلى :

1 - أيقونة.

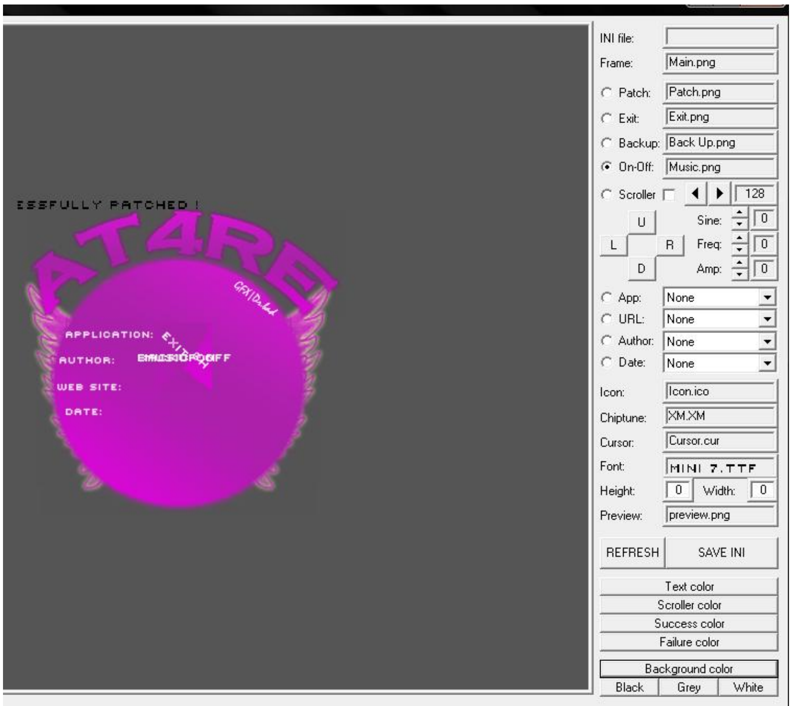
2 - ملف الخط المراد استخدامه (يجب أن يكون صغير الحجم؛ لأنه يؤثر على حجم الباش الكلي).

3 - ملف نشيد من نوع xm (هذه الخاصية اختيارية يمكن الاستغناء عنها).

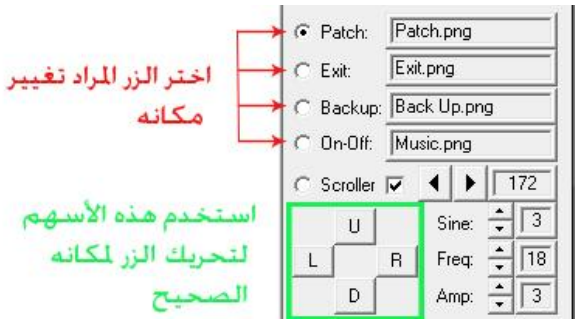
4 - مؤشر الفأرة: البرنامج يدعم النوعين: النوع الثابت (CUR.) والنوع المتحرك (ANI.) (هذه الخاصية اختيارية يمكن الاستغناء عنها).

4. الخطوة الأخيرة في تصميم السكين هي إنشاء ملف (ini.) وهو الملف المسؤول عن تجميع السكين وأماكن الأزرار والمعلومات التي تحتويها الباش؛ كاسم البرنامج وغيره، وسنستخدم لإنشائه برنامج uPPP SkinHelper v0.1 .

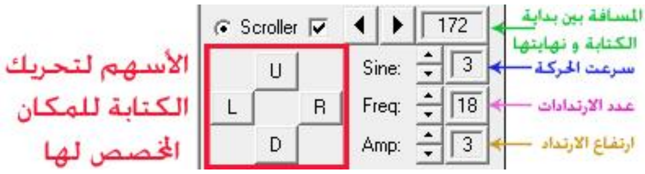
1 - قم بسحب الصور إلى أماكنها المخصصة في البرنامج، كما هو موضح في الصورة، وكذلك الأمر بالنسبة للأيقونة والخط وملف النشيد :



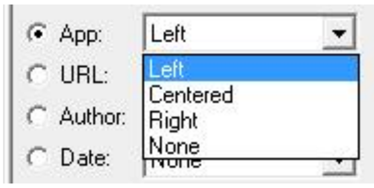
2 - اختر الزر الذي تريد تغيير مكانه، واستخدم الأسهم لتغيير المكان :



3 - الآن عليك تغيير مكان الكتابة المتحركة، وتعديل خصائصها كما تريد :



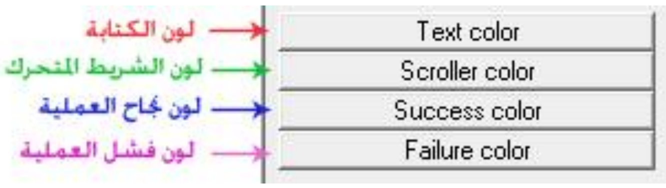
4 - نقوم الآن بتغير أماكن المعلومات كما قمنا بتغيير أماكن الأزرار (بالنسبة للمعلومة التي تريد الاستغناء عنها اترك الخيار None) واستخدم الأسهم أيضاً لوضعها في مكانها المناسب :



5 - قم باختيار حجم الخط الذي تريد :



6 - غيّر الألوان بما يتناسب مع السكين الذي قمت بتصميمه :

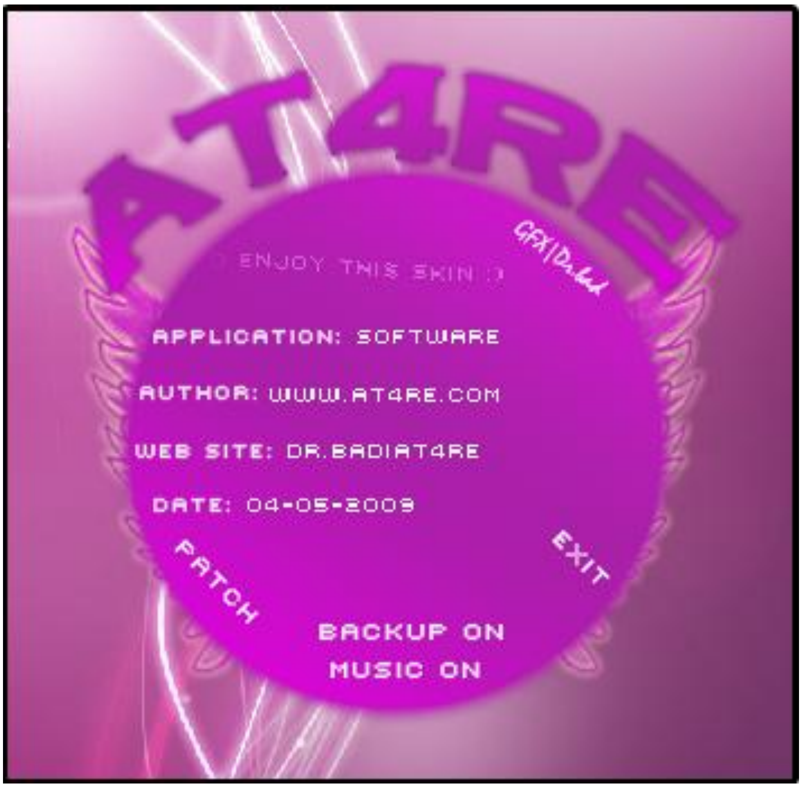


7 - قم بحفظ الملف عن طريق الزر، ويمكنك أيضاً استخدام زر (REFRESH) لتشاهد التغييرات أمامك :



5. الآن وبعد الانتهاء من التصميم يمكنك استخدام السكين في برنامج Uppp وإنشاء الملف التنفيذي لمشروعك.

2. النتيجة



تصميم سكين لبرنامج Uppp باستخدام RealDraw 5



برنامج RealDraw 5 أو كما سنسميه RD5 يمكن أن نقول: إنه يحتل المرتبة 2 بعد برنامج الــ Photoshop الشهير، من حيث برامج التصميم, في درسنا هذا سنقوم بتصميم سكين Uppp (Transparent Skin) بهذا البرنامج.

- Real Draw.
- صورة الزهرية بالمرفقات.

(التحميل من مرفقات الكتاب).

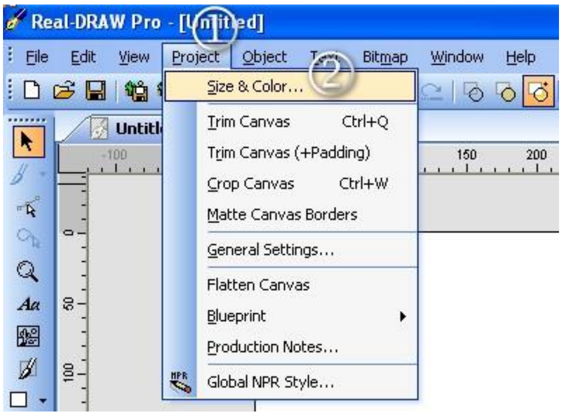
تعمدنا جعل هذا السكين هو الدرس، وذلك راجع لاستعمال الكثير من ميزات وأدوات البرنامج من أجل الإلمام به في درس واحد.

لنبدأ على بركة الله :

1. Rainbow I Uppp Skin

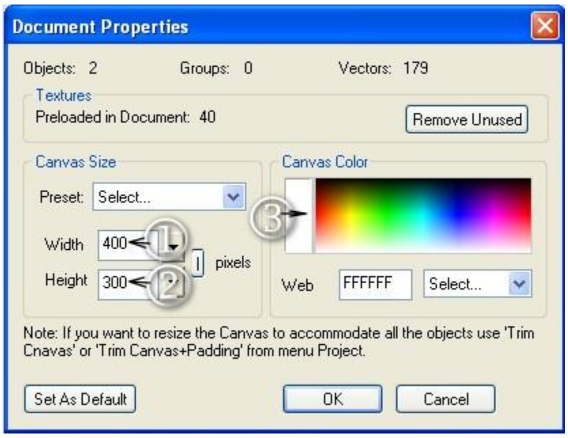
إعداد البرنامج:

بعد فتح البرنامج :



1) اختر Project

2) بعدها Size & Color...

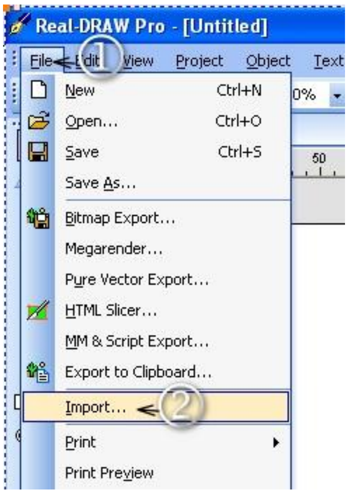


1) عرض التصميم بالبكسل.

2) ارتفاع التصميم بالبكسل.

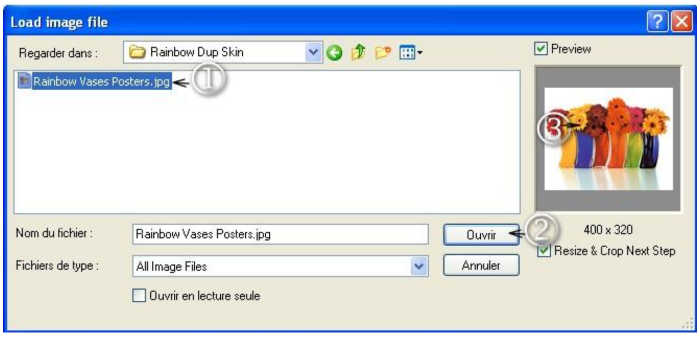
3) لون خلفية التصميم.

انتهينا من الإعداد، والآن سنستورد الصورة المراد العمل عليها :

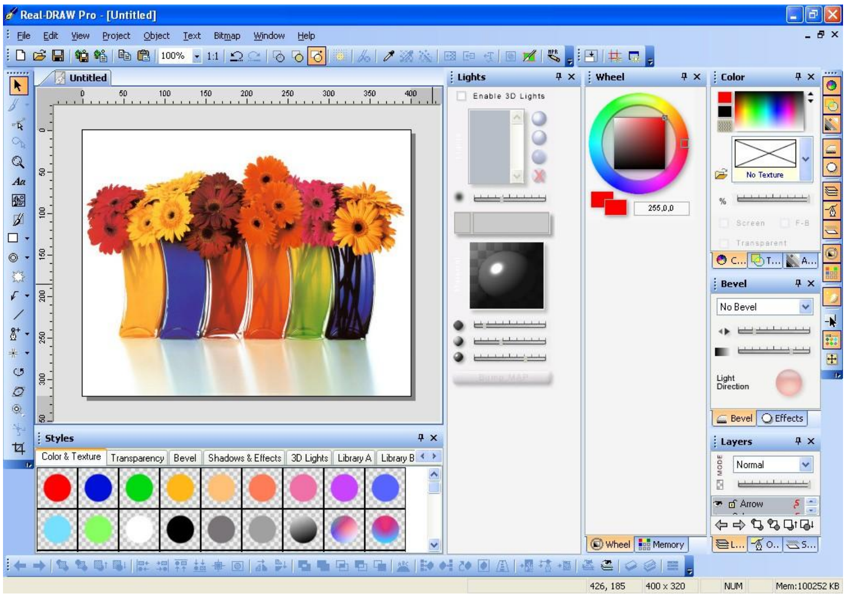


1) اختر File

2) بعدها Import...



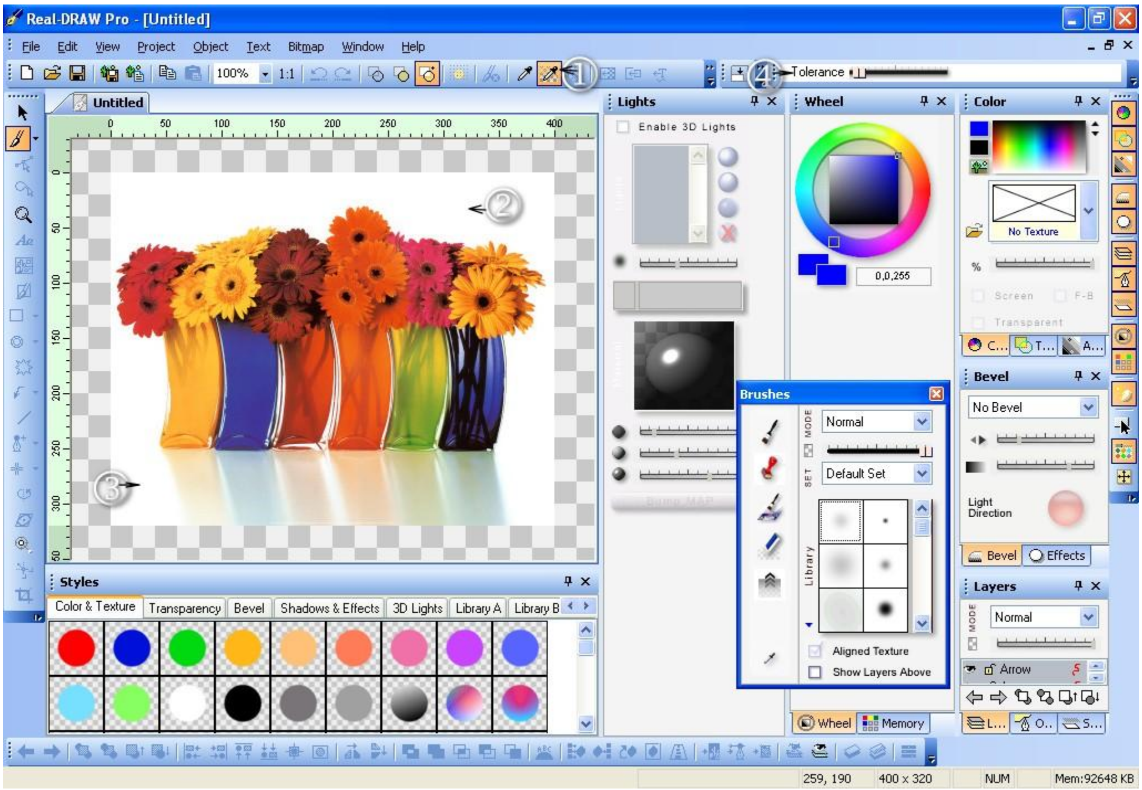
واجهة البرنامج بعد استيراد الصورة :



كما ترى لدينا 6 مزهريات، وتراودنا فكرة أن نجعل شعار الفريق "AT4RE" كل حرف من هذه الــحروف الخمسة على مزهرية من المزهريات.

علينا الاستغناء عن واحدة, وأيضا جعل السكين شفافة في إزالة لون الخلفية الأبيض.

قم بالضغط مرتين على الصورة بالزر الأيسر للفأرة، ستنفتح نافذة التعديل كما يلي :



1)  هذه الأداة تدعى شفافية اللون (Color Transparency) والغرض منها إزالة الألوان الغير مرغوب بها في مناطق معينة من التصميم / الصورة. ولأن السكين المراد عمله شفاف؛ لذا سنزيل اللون الأبيض للخلفية.

2) بعد اختيار الأداة 1 قم بالضغط في الموقع 2 .

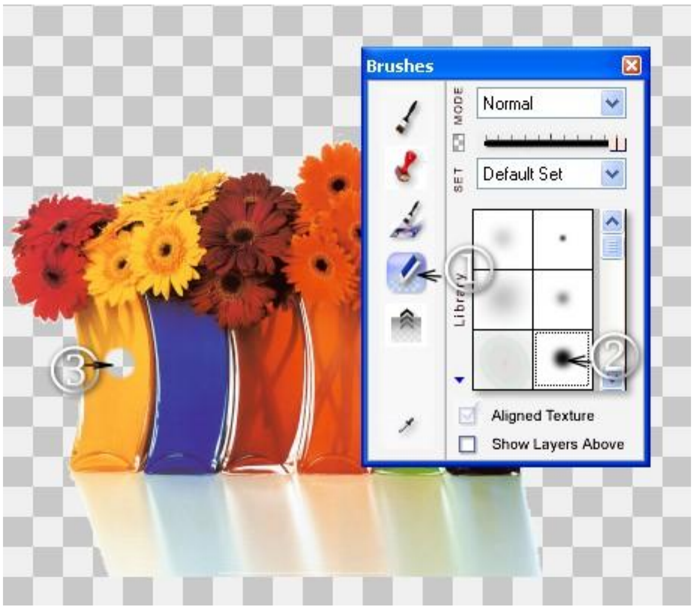
3) وكذلك في الموقع 3 لإزالة لون الخلفية الأبيض الموضح في الصورة أعلاه.

4) هذه الأداة مهمتها تتمثل في شمولية الألوان المراد اقتصاصها، على سبيل المثال إذا كانت المنطقة محتوية على أكثر من لون واحد فعلينا تحريك هذه الأداة لليمين بقدر معين على حسب عدد الألوان الموجودة بتلك المنطقة، ففي مثالنا هذا لا نحتاج لتحريكها؛ لأن لون الخلفية واحد فقط، وهو اللون الأبيض :

النتيجة :

قبل:	بعد:
	

الآن لنزيل المزهرية الصفراء الموجودة باليسار:



1) أداة الممحاة لإزالة المناطق الغير مرغوب فيها.

2) حجم الـــ Brush أو رأس الممحاة.

3) كما ترى بعد الضغط بالزر الأيسر للفارة في المكان 3 تم إزالة اللون البرتقالي منه، وبقيت المنطقة شفافة.

ماعليك الآن إلا إكمال باقي المزهرية.

النتيجة :



كما تلاحظ شكل المزهرية تحسن ولكن توجد به عيوب :

1) اللون الأزرق والأحمر مكرران.

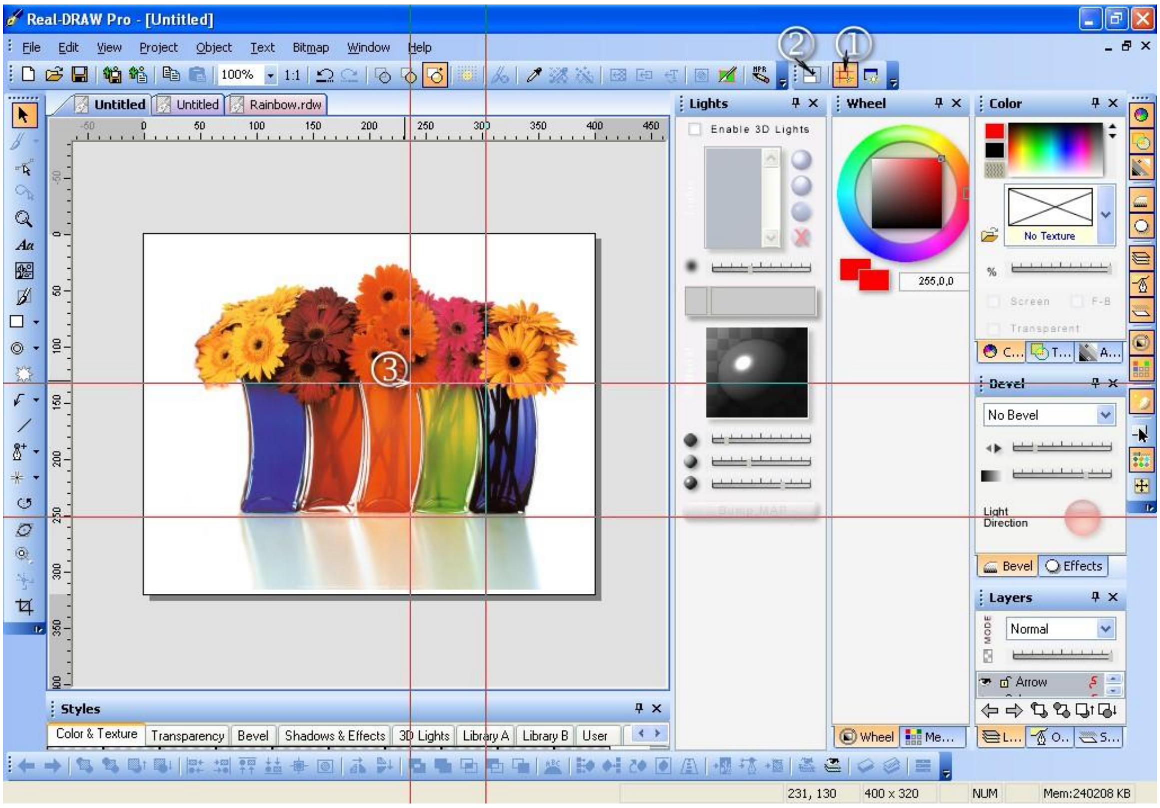
2) المزهرية الأولى لونها غير ملائم مع تظليل داخلي سيئ جداً.

3) المزهرية الزرقاء الأخيرة غير شفافة.

لنعمل على إصلاح هذه الأشياء :

إزالة تكرار ألوان المزهريتان – الزرقاء والحمراء- بإنشاء مزهريتين بلونين آخرين :

الفكرة تتمثل في التقاط صورة عن مزهرية تبدو جيدة من ناحية الشفافية واللون، كالمزهرية الخضراء، وبعد ذلك تغيير لونها :



1) الأداة 1 لالتقاط صورة لمكان معين من الشاشة.

2) الأداة 2 لتصغير البرنامج، والغرض منه التقاط صورة لمكان آخر من غير البرنامج كــ سطح المكتب.

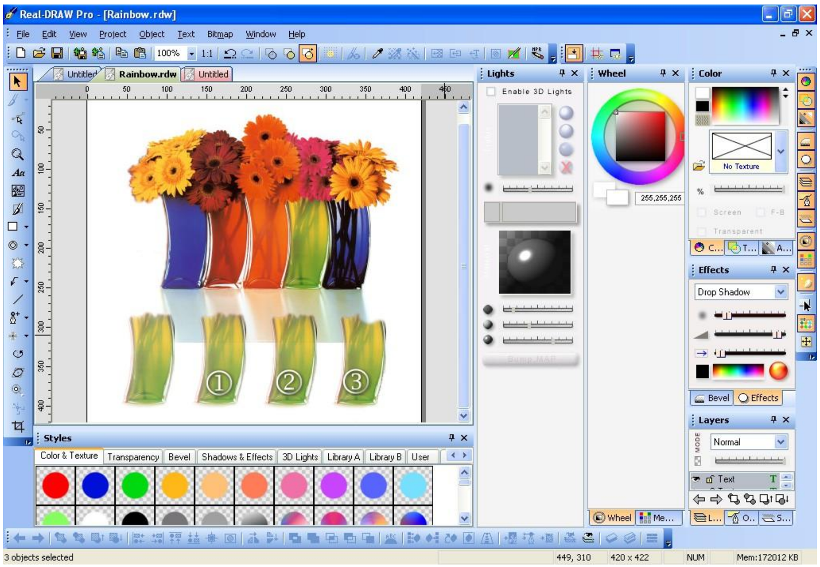
1) قم بإزالة التعيين عنها؛ لأننا نريد عمل Screenshot من واجهة البرنامج للمزهرية الخضراء.

2) نضغط بالزر الأيسر للفأرة في المكان 3 لبدء تحديد المكان المراد أخذ Screenshot له.

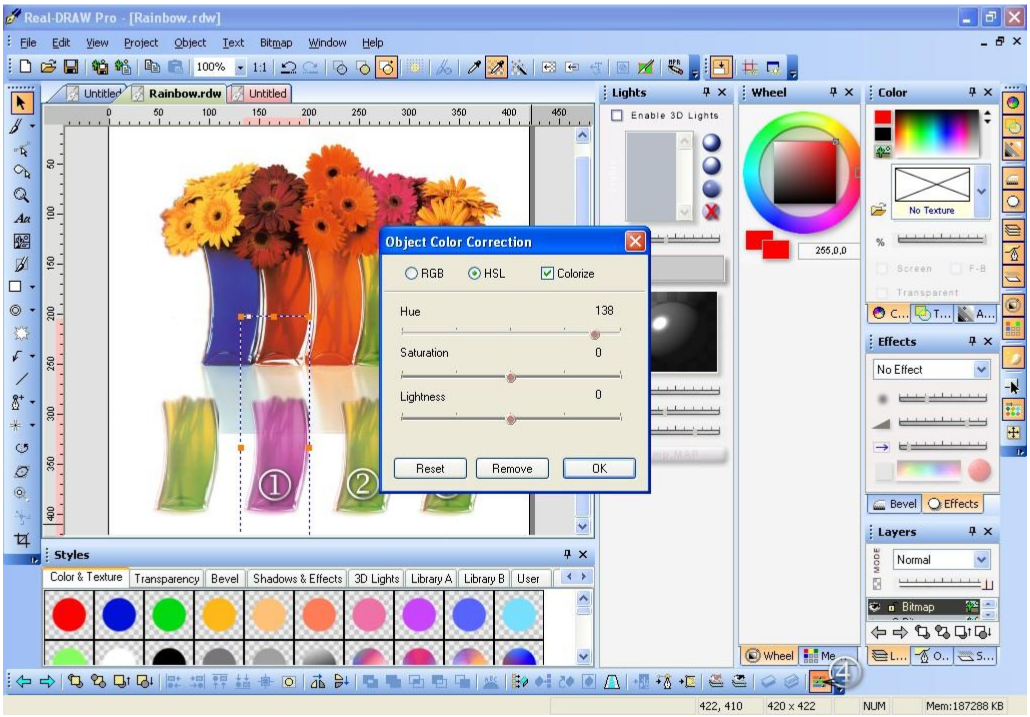
بعد ذلك قم بإزالة المناطق المحيطة :

قبل:	بعد:
	

نقوم بإنشاء 3 نسخ من قالب المزهرية الخضراء :



في الصورة التالية: بالأداة 4 نقوم بتغيير لون النسخة 1 إلى لون جيد ومناسب :



نفس الشيء للنسخ 2 و3 :



بعد وضع النسخ فوق المزهريات المراد تبديل لونها :

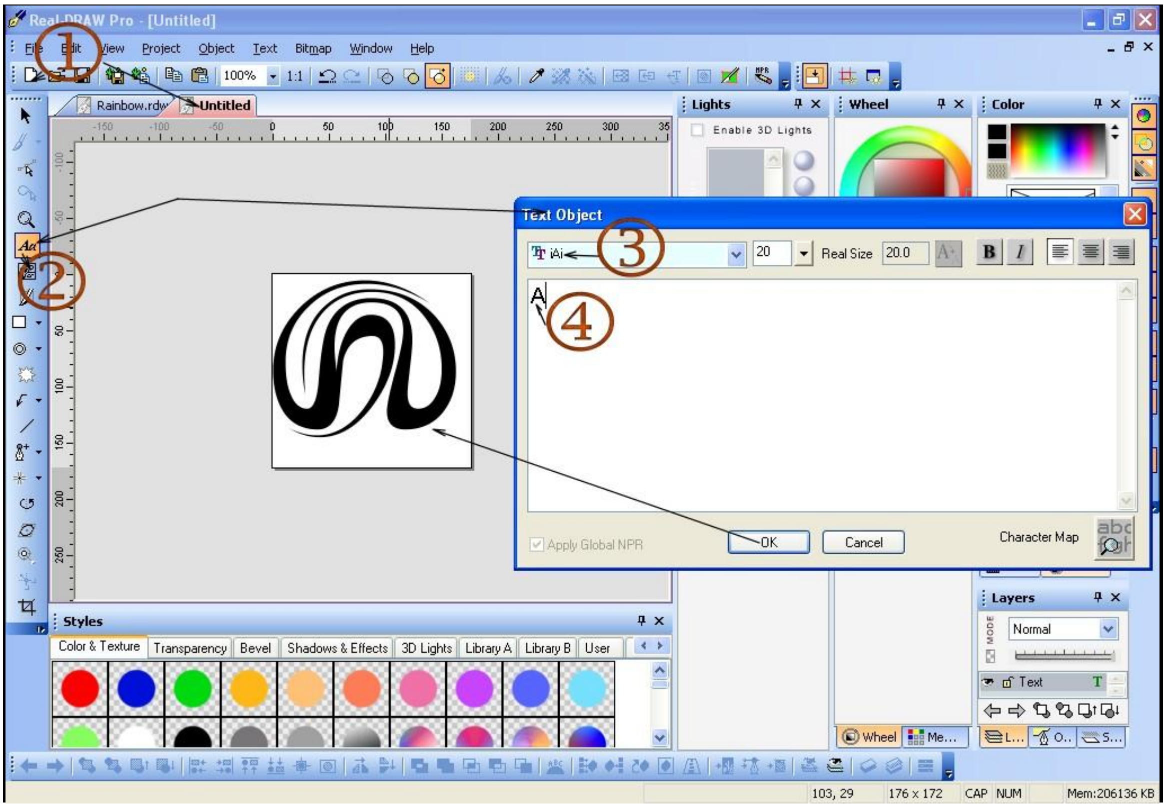


الظل أزرق والمزهرية بنفسجية؟!

بسيطة، نعمل نفس الشيء كما عملنا للنسخ الماضية: نسخ, إزالة، شفافية، وتبديل لون :



2. شعار الفريق



1) لإنشاء صفحة / مستند جديد.

2) أداة الكتابة.

3) اختيار خط الكتابة المناسب.

4) مكان الكتابة.

نكتب A ثم نقوم بنسخها، بعدها نقوم بلصقها فوق السكين، ونضعها فوق المزهرية الأخيرة كما يلي :

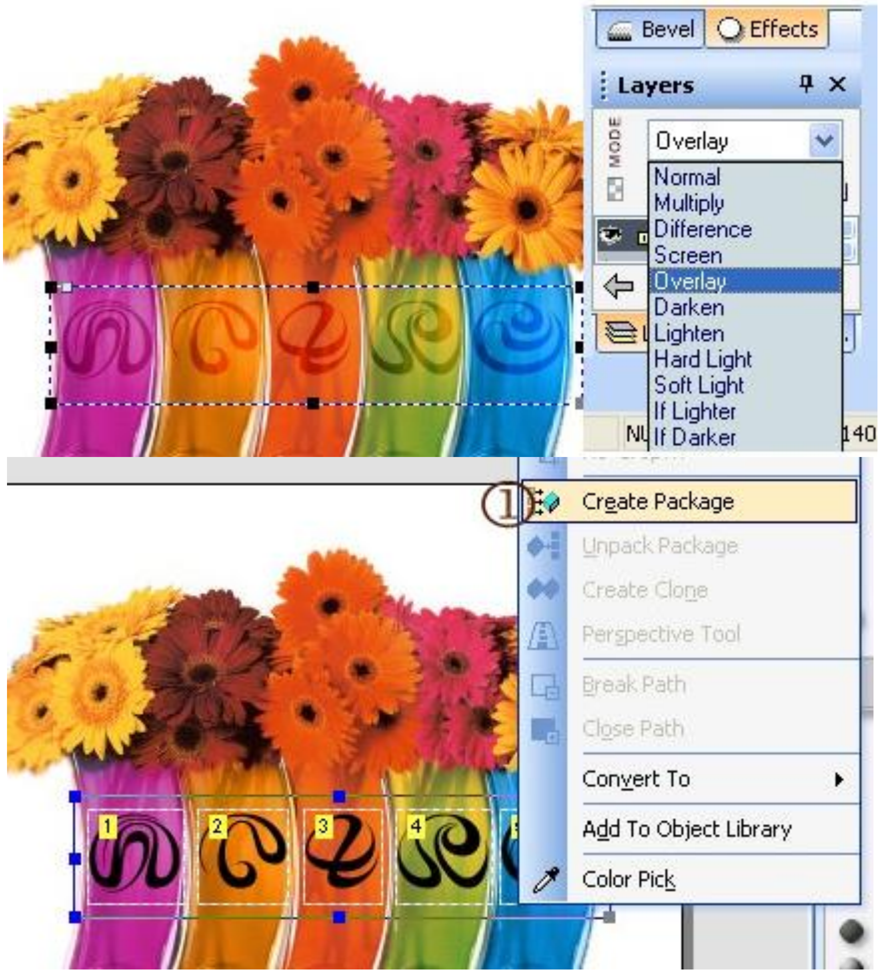


نعمل نفس الحركة لباقي الحروف :



نقوم الآن بمجانسة الكتابة فوق المزهريات لتبدوا أفضل.

نقوم بتحديد جميع الحروف، ونضغط بالزر الأيسر للفأرة، ونختر 1, والذهاب لـــ Layers واختيار Overlay :



a

باستعمال Poly-Line : "الخط الأحمر" وإضافة أزرار نتحصل على :



بالنسبة لتصميم الأزرار تركناها لك، أما كيفية استعمال السكين مع برنامج Uppp فقد تكرم الأخ Dr.bad بشرح وافٍ لذلك.

3. تصميم سكين لبرنامج Dup باستخدام RealDraw 5

قمنا بتضمين السورس للتعلم منه :

Rainbow II Dup Skin



الباب التاسع: متفرقات

إتقان أدوات الهندسة العكسية

Mastering RCE tools

RadAsm, OllyDbg and Imprec as examples

سيكون درس اليوم خليطاً مما يمكن للمهندس العكسي المبتدئ أن يلاقيه أثناء عمله، وهذا ما سنتعلمه :

1 - تقليد برمجة برنامج صغير.
2 - تقليد إنشاء حماية للبرنامج.
3 - استعمال أمثل للمنقح OllyDbg، لإصلاح أخطاء في البرنامج.
4 - عكس هندسة البرنامج للتخلص من الحماية.

و لنبدأ باسم الله :

جميع ما تحتاجه في المرفقات.

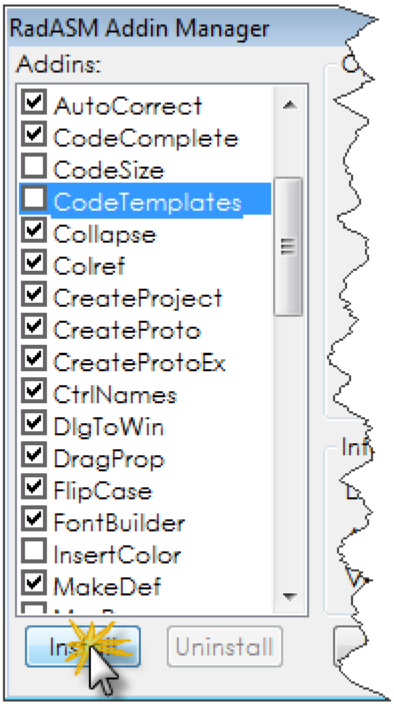
I. كتابة البرنامج

لن نركز هنا عن كتابة البرنامج، إنما عن استعمال RADASM العتيد بكل ما فيه من ترسانة تسهل كتابة برامج الأسمبلي، وسيكون البرنامج هنا عبارة عن مترجم بسيط جداً يحول سلسلة محارف من الصيغة المتتابعة إلى صيغة مصفوفة متناسبة مع الأسمبلي.

I.1. بعض الإعدادات المهمة قبل البدء

سنحتاج إلى تثبيت وتفعيل بعض الإضافات الهامة لـــ RADASM وستجدها في المجلد Extra/RadPlugins والطريقة سهلة جداً:

1 - استخرج جميع الملفات إلى المسار %RADASM%\AddIns إذا لم تكن هذه الإضافات موجودة فيه بالأصل.
2 - افتح RADASM واذهب إلى القائمة : Options→addin manager ثم فعّل خانات هذه الإضافات، وفي كل مرة انقر على Install أو فقط انقر مرتين على كل إضافة غير مفعلة.



و الإضافات هي :

1 - AutoCorrect : إضافة تسمح بتدارك الأخطاء وتسريع كبير للكتابة.

2 - CodeTemplates : أيضاً إضافة رائعة لتسريع الكتابة بإضافة الـ snippets من الكود.

3 - DlgToWin : أيضاً لتسريع كتابة الكود، وتحويل ملف الـ dlg (ملف الـ dialog المعتمد من صاحب RADASM إلى كود مبني على الـ API).

4 - VersionControl : به العديد من المزايا, فبالإضافة إلى إمكانيته في عمل incrementation لنسخة البرنامج, فهو يمكنك من حفظ نسخ احتياطية من الملفات المصدرية لبرنامجك وإمكانية استعادتها.

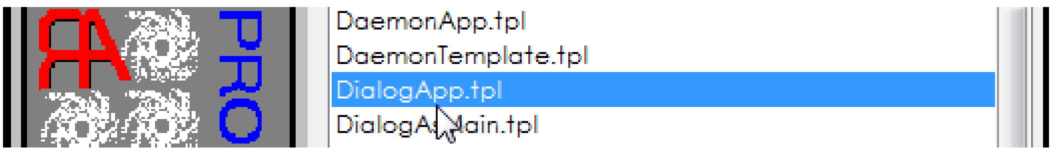
5 - TxtLnkMan : رائع لتتبع الكود وإضافة التعليقات وTODO LIST.

6 - TabAddin : تنسيق وترتيب الكود.

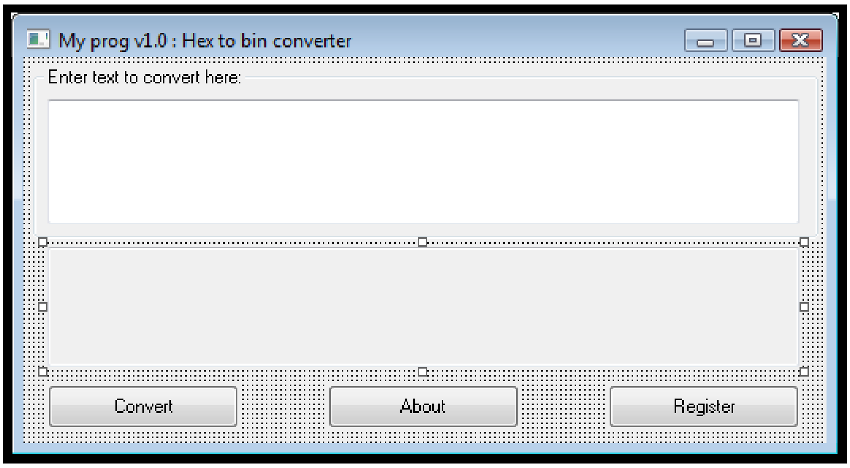
إذن بعد أن تفعّل هذه الإضافات (إن لم تكن مفعّلة مسبقاً) انتقل إلى بدء كتابة الكود.

I.2 كتابة الكود

أنشئ مشروعاً جديداً كالمعتاد واختر كـ Template المسماة Dialog as app (فقط لإضافة الـ controls بسهولة).



بعد ذلك أضف أزراراً وأماكن كتابةٍ كما في المثال:

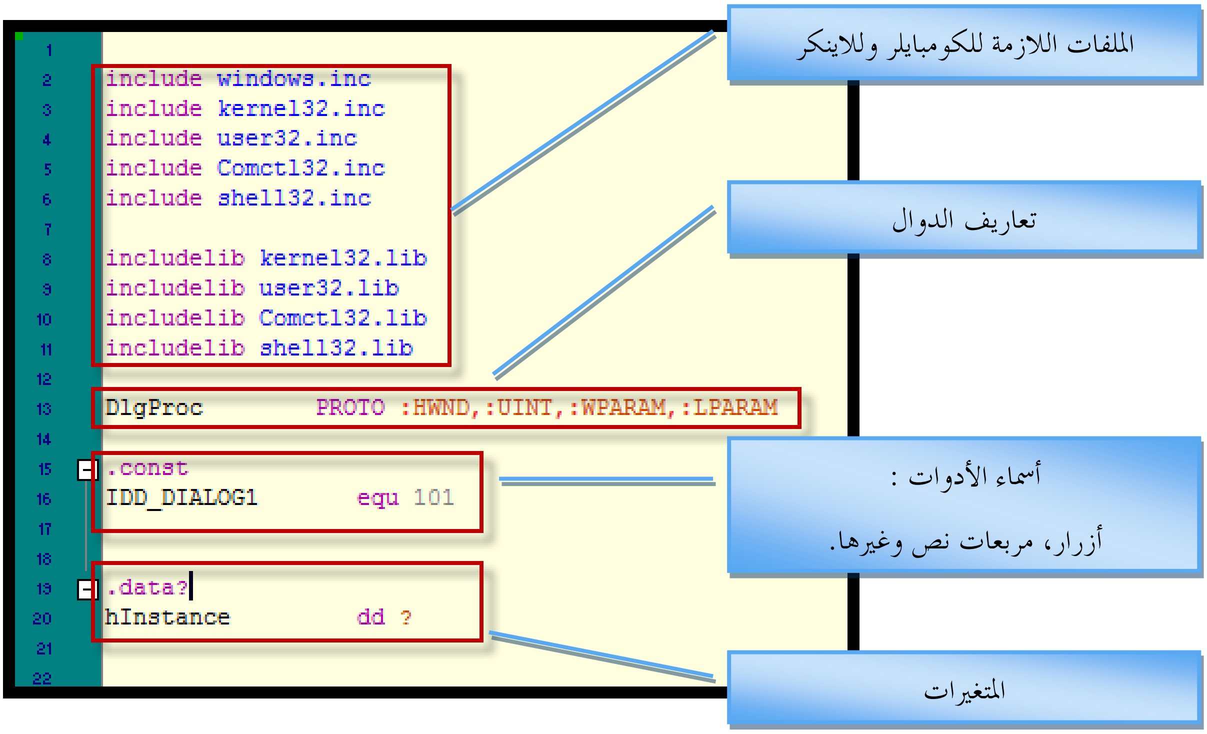


وغيّر الأسماء الافتراضية إلى :

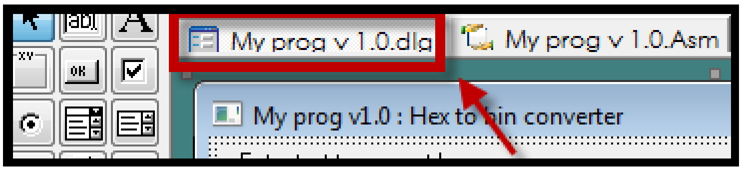
IDD_DIALOG1	: اسم النافذة الأم	BTN_CONV	:
EDT_TEXT_IN	: اسم الـــ Editbox العليا	BTN_ABOUT	: أسماء الأزرار.
EDT_TEXT_OUT	: اسم السفلى	BTN_REG	:

وهذه التسمية مفيدة جداً, فهي ستمكنك من التعامل مع أسماء سهلة التذكر بدل التعامل مع أعداد تضطر إلى الرجوع إليها في كل مرة.

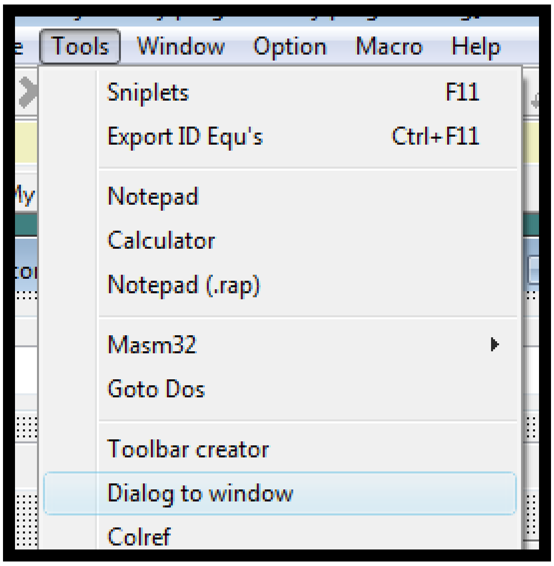
و الآن, اذهب إلى الملف inc لإضافة هذه الأسماء في الفقرة const.

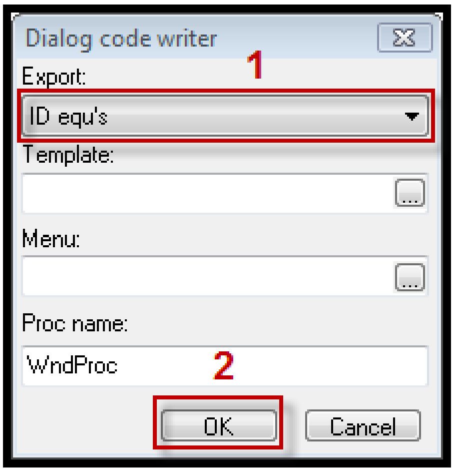


ثم اذهب إلى اللسان الخاص بالــ Dialog :



ثم إلى القائمة DialogToWin → Tools (CTRL+F11 لاستخدام إضافة أخرى هي ResourceID)





بعدها انقل الكود الناتج إلى الفقرة const. لينتج الشكل التالي :

```
.const
;My prog v 1.0.dlg
IDD_DIALOG1                    equ 101
GRP_1                          equ 1006
EDT_TEXT_IN                    equ 1001
EDT_TEXT_OUT                   equ 1002
BTN_CONV                       equ 1003
BTN_ABOUT                      equ 1004
BTN_REG                        equ 1005

;Res\My prog v 1.0Ver.rc
VERINF1                          equ 1
```

ثم اذهب مرة أخرى DialogToWin → Tools واختر هذه كالتالي :



و الآن سترى كيف ستسهل عليك الأمور لو كان لديك 10 أزرار أو أكثر. وإليك الناتج بعد وضعه في المكان المناسب :

قبل التعديل

```
DlgProc proc hWin:HWND,uMsg:

    mov     eax,uMsg
    .if eax==WM_INITDIALOG

    .elseif eax==WM_COMMAND

    .elseif eax==WM_CLOSE

    .else
        mov     eax,FALSE
        ret
    .endif
    mov     eax,TRUE
    ret

DlgProc endp
```

بعد التعديل

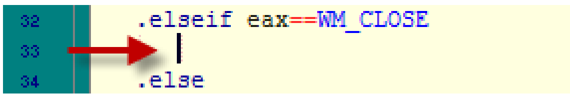
```
DlgProc proc hWin:HWND,uMsg:UINT,

    mov     eax,uMsg
    .if eax==WM_INITDIALOG

    .elseif eax==WM_COMMAND
        mov eax, wParam
        and eax, -1
        .if eax==BTN_CONV
        .elseif eax==BTN_ABOUT
        .elseif eax==BTN_REG
        .endif
    .elseif eax==WM_CLOSE

    .else
        mov     eax,FALSE
        ret
    .endif
    mov     eax,TRUE
    ret

DlgProc endp
```

ثم اذهب تحت WM_CLOSE () ثم اضغط CTRL+J واختر « Dialog end ».

أكيد لن تجدها، لكن يمكنك أن تضيف ما تشاء من سطور تكررها عادة إلى الملف « masm.ct » والذي تجده في المسار %RADASM%\Addins وذلك ليكون ما تضيفه على الشكل، مع ذكر أن الإضافة المسؤولة هي CodeTemplates :

1. الاسم الذي يظهر.
2. ما تريده أن يكتب من سطور حسب ما تريد.
3. ...
4. ...
5. الاسم الثاني.
6. السطور التي تريدها أن تعوض.

و هذا مثال ستحتاجه كثيراً (سنحتاجه أيضاً في هذا الدرس إن شاء الله).

افتح الملف المذكور وقم بعمل نسخ للدالة DlgProc كاملة، واحذف ما يتغير عادة وأعط الكل اسماً :

```
DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

	mov		eax,uMsg
	.if eax==WM_INITDIALOG

	.elseif eax==WM_COMMAND
		mov eax, wParam
		and eax, -1
		.if eax==BTN_1
			~
		.elseif eax==BTN_2

		.elseif eax==BTN_3

		.endif
	.elseif eax==WM_CLOSE
		invoke EndDialog, hWin, 0
	.else
		mov		eax,FALSE
		ret
	.endif
	mov		eax,TRUE
	ret

DlgProc endp
```

السهم يشير إلى الإشارة ~ التي يجب إضافتها في المكان الذي تريد أن يكون فيه مؤشر الفأرة، واحفظ الملف ثم اعد تشغيل RADASM ليكون الشكل النهائي كالتالي :

```
{Dialog end}
invoke EndDialog, hWin, 0
~
{Dialog Procedure}
DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

    mov     eax,uMsg
    .if eax==WM_INITDIALOG

    .elseif eax==WM_COMMAND
       mov eax, wParam
       and eax, -1
      .if eax==BTN_1
          ~
      .elseif eax==BTN_2

      .elseif eax==BTN_3

      .endif
    .elseif eax==WM_CLOSE
      invoke EndDialog, hWin, 0
    .else
        mov     eax,FALSE
        ret
    .endif
    mov     eax,TRUE
    ret

DlgProc endp
```

أضف هذا السطر لإضافة إحدى أهم وأشمل المكتبات التي لا يمكن الاستغناء عنها. وهي تأتي مع MASM ولكن أرفقناها مع ذلك (تجدها في Extra\Libs)

```
include user32.inc
include Comctl32.inc
include shell32.inc

includelib kernel32.lib
includelib user32.lib
includelib Comctl32.lib
includelib shell32.lib

include macros.asm

uselib masm32,cryptohash
```

```
include masm32.inc
includelib masm32.lib
include cryptohash.inc
includelib cryptohash.lib
```

الكلمة المفتاح userlib هي عبارة عن ماكرو تجده في الملف macros.asm .

و السطر السابق يساوي ما على اليسار من سطور.

الآن, حاول أن تعمل Compiling للمشروع وستلاحظ بعض الأخطاء، وتتمثل في :

```
:ime 172 ms
 \Bin\ML.EXE /c /coff /Cp /nologo /I"[redacted]\Include" "My prog v 1.0.asm"
/ prog v 1.0.asm
inc(15) : fatal error A1000: cannot open file : \masm32\include\masm32.inc
acro Called From
```

و تصحيح هذا سهل, ما عليك إلا أن تفتح الملف macros.asm وتبحث عن الماكرو userlib وتصلحها هكذا : (انتبه : اترك مسافة قبل >)

```
uselib MACRO args:VARARG
  LOCAL acnt,buffer,var,lbl,libb,incc,buf1,buf2
  acnt = argcount(args)
  incc equu <include >
  libb equ <includelib >
  var = 1
:lbl
  buffer equ getarg(var,args)
```

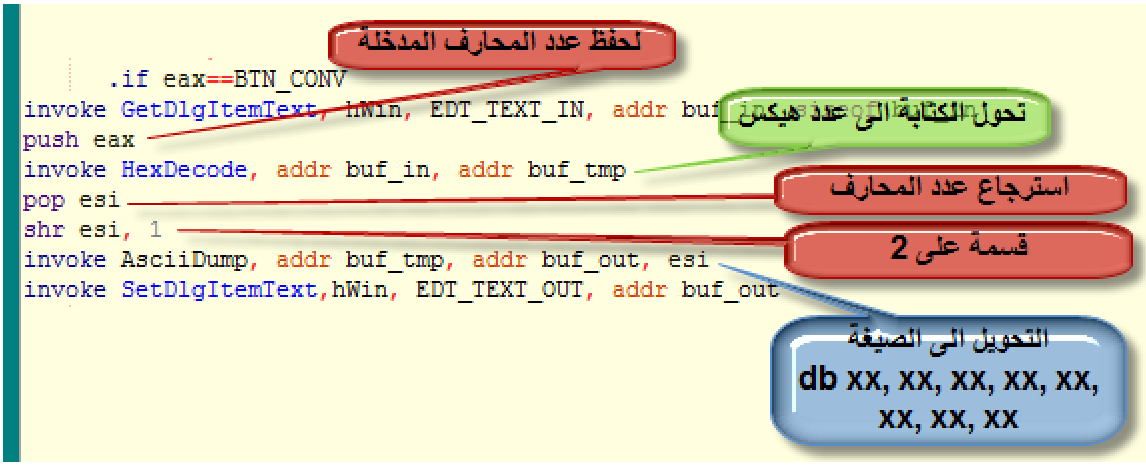
كل شيء أصبح على أتم وجه. فلنبدأ في إضافة الكود اللازم لعمل برنامجنا :

```
    .if eax==BTN_CONV
      invoke GetDlgItemText, hWin, EDT_
    .elseif eax==BTN_ABOUT
    .elseif eax==BTN_REG
    .endif
.elseif eax==WM_CLOSE
    invoke EndDialog, hWin, 0
.else
    mov     eax,FALSE
    ret
.endif
mov     eax,TRUE
ret
```

EDT_TEXT_IN
EDT_TEXT_OUT

تتذكر طبعاً الأسماء التي أسمينا بها الـــ Controls الخاص بالفورم, لا بأس إذا نسيتها ما عليك إلا الضغط على CTRL+SPACE بعد كتابة بعض الأحرف التي تتذكرها كـــ _EDT وRADASM وسيعطيك جميع الأسماء المبتدئة بـــ هذه الأحرف لتختار منها.

نتمم بكتابة ما يلي :



بالأخضر : دالة من المكتبة cryptohash.lib وتقوم بتحويل الكتابة إلى عدد ستة عشري.

وبالأزرق : دالة من المكتبة masm32.lib سرتقوم بكل ما تحتاجه، وهي معرفة في ملف المساعدة التابع لها بالشكل التالي :

1- عنوان Buffer سلسلة البايتات المراد فرمتتها.

2- عنوان Buffer المتلقي.

3- طول السلسلة.

Parameters

1. **lpsrc** The address of the byte data to be converted.
2. **lpbuf** The address of the buffer to receive the results.
3. **lnsrc** The length of the byte data to be converted.

Return Value

There is no return value.

و هذا الكود كاملاً :

```
DlgProc proc hWin:HWND,uMsg:UINT,wParam:WPARAM,lParam:LPARAM

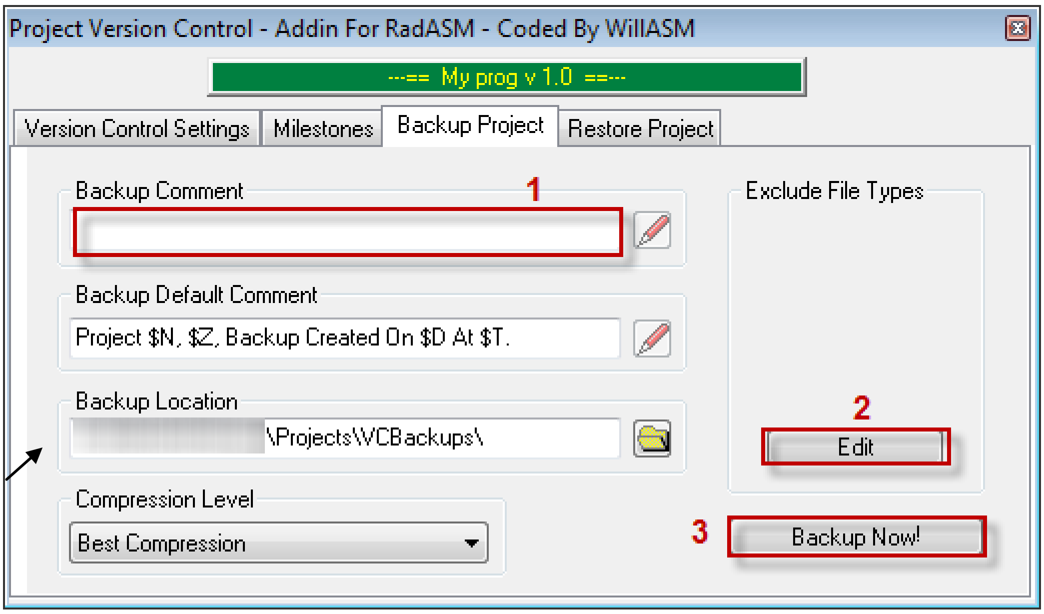
    LOCAL hFile:DWORD
    LOCAL fSize:DWORD
    LOCAL buf_in[MAX_PATH]:BYTE
    LOCAL buf_out[MAX_PATH]:BYTE
    LOCAL buf_tmp[MAX_PATH]:BYTE

    mov      eax,uMsg
    .if eax==WM_INITDIALOG

    .elseif eax==WM_COMMAND
        mov eax, wParam
        and eax, -1
        .if eax==BTN_CONV
invoke GetDlgItemText, hWin, EDT_TEXT_IN, addr buf_in, sizeof buf_in
push eax
invoke HexDecode, addr buf_in, addr buf_tmp
pop esi
shr esi, 1
invoke AsciiDump, addr buf_tmp, addr buf_out, esi
invoke SetDlgItemText,hWin, EDT_TEXT_OUT, addr buf_out
        .elseif eax==BTN_ABOUT
        .elseif eax==BTN_REG
        .endif
    .elseif eax==WM_CLOSE
        invoke EndDialog, hWin, 0
    .else
        mov      eax,FALSE
        ret
    .endif
    mov      eax,TRUE
    ret

DlgProc endp
```

فلنقم بعمل backup للمشروع : اذهب إلى Project→Version contorl→Backup project tab



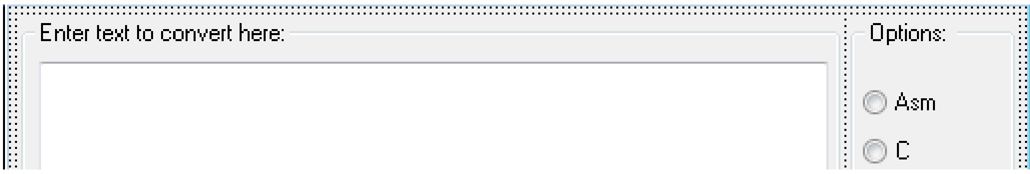
في 1: أدخل تعليقاً لك؛ حتى تفرق بين النسخ المحفوظة المختلفة.

في 2: أدخل extensions التي لا تريدها أن تنسخ مع الملفات المحفوظة: غالباً exe, obj, res, ilk و idb .

في 3: اضغط لحفظ الملف على صيغة zip في المسار (السهم) الذي اخترته.

(اذهب إلى الألسن الأخرى للتعرف على خاصيات هذه الإضافة).

و الآن سنضيف بعض الكود للاختيار بين الصيغة الموافقة للأسمبلي والموافقة للـــ C ليصبح الشكل هكذا :

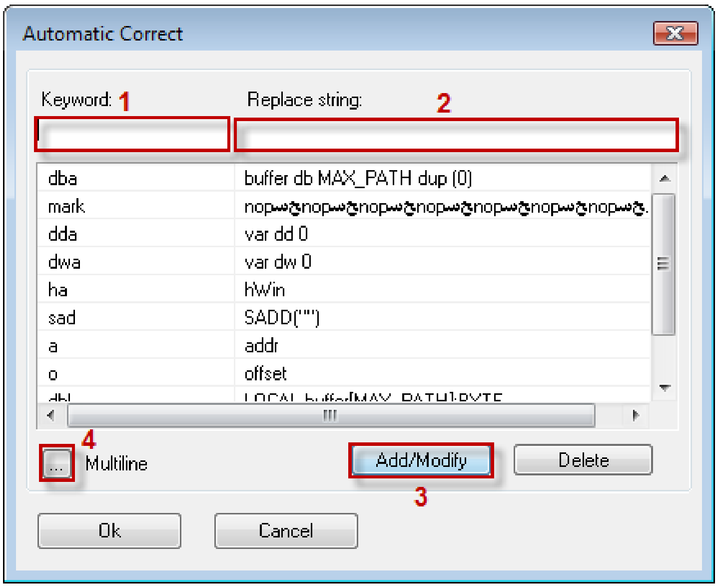


ثم أضف الـــ IDs مع إعادة التسمية :



و الآن أضف الكود كما في الصورة التالية : (لاحظ أن الكود القديم تم تعتيمه والإبقاء على الكود الجديد).

أثناء كتابتك لهذه الأسطر, ستلاحظ أن العديد من الكلمات والأسطر تتكرر بشكل دائم, والحل سهل لعدم كتابتها حرفياً, ما عليك إلا الذهاب إلى Tools→Auto correct list وإدخال الحرف أو الكلمة التي تريدها أن تعوض الكلمة التي ستدخلها على اليسار :



اكتب في 1 الحرف أو الكلمة التي تريدها أن تعوض ما في 2 ثم احفظ بالضغط على 3.

وإذا كان ما تريده في 2 متكون من العديد من الأسطر, أدخل تلك الأسطر في 4.

و هكذا لكتابة addr مثلاً, يكفي كتابة a والضغط على SPACE ليكمل البلاغين الكلمة.

الآن هنا الكود الكامل بعد التعديل :

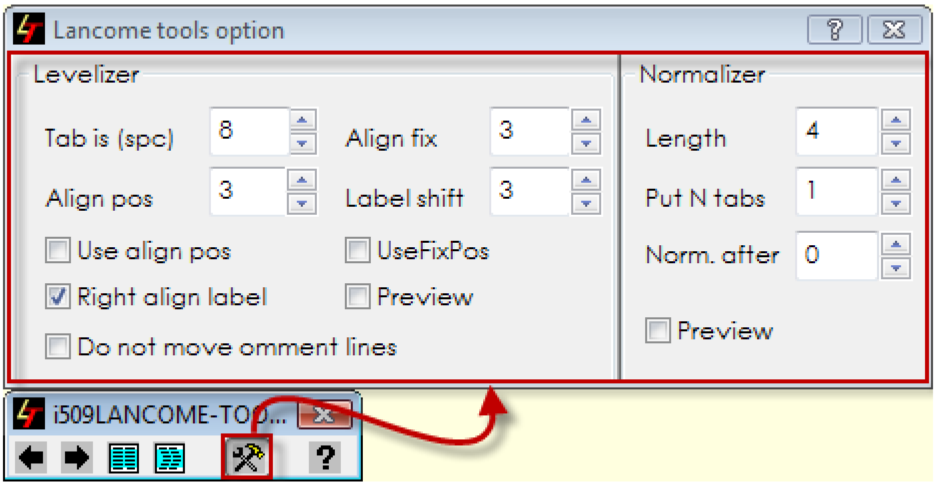
```
    mov      eax,uMsg
    .if eax==WM_INITDIALOG
   invoke CheckRadioButton, hWin, RBN_ASM, RBN_C, RBN_ASM
    .elseif eax==WM_COMMAND
       mov eax, wParam
       and eax, -1
       .if eax==BTN_CONV
invoke GetDlgItemText, hWin, EDT_TEXT_IN, addr buf_in, sizeof buf_in
push eax
invoke IsDlgButtonChecked, hWin, RBN_ASM
.if eax
invoke HexDecode, addr buf_in, addr buf_tmp
pop esi
shr esi, 1
invoke AsciiDump, addr buf_tmp, addr buf_out, esi

.else
   invoke lstrcpy, addr buf_out, SADD("char array[] = {") ; copy array start
   pop esi
   xor ecx, ecx
   mov edi, 16
   .while ecx < esi
mov ax, word ptr buf_in[ecx]
mov word ptr buf_out[edi], 'x0' ; concatenate hex prefix for c/c++
mov word ptr buf_out[edi+2], ax  ; concatenate the enetered byte
mov word ptr buf_out[edi+4], ' ,'   ; concatenate separator
add ecx, 2
   add edi, 6
   .endw
   mov word ptr buf_out[edi-2], ';}'   ; concatenate array end operator
   mov word ptr buf_out[edi], 0A0Dh ; (return)
   mov byte ptr buf_out[edi+2], 0 ; (null terminator)

.endif
invoke SetDlgItemText,hWin, EDT_TEXT_OUT, addr buf_out
       .elseif eax==BTN_ABOUT
       .elseif eax==BTN_REG
       .endif
```

إذا رجعت إلى صورة السورس السابقة ستجد أن الكود يبدو بشعاً جداً؛ لتصفيفه بسرعة اذهب إلى View→Lancome tools

وستظهر لك نافذة بها عدد من الأزرار يمكنك ترك الإعدادات كما هي أو تغييرها كما تريد (عليك بالتجريب) :



اعمل اختيار الكل (CTRL+A) واضغط الزر الأسفل لترتيب الكود :

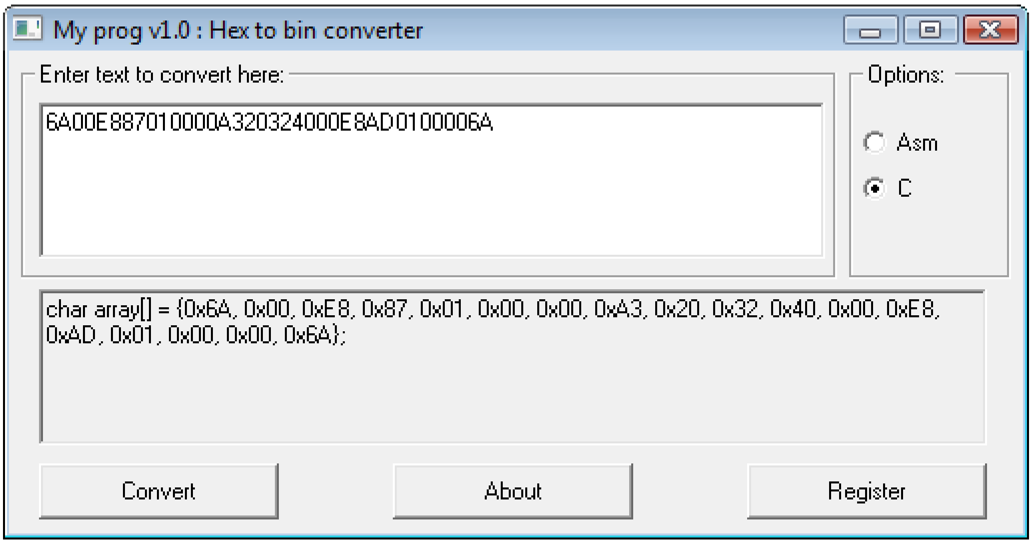


وإليك النتيجة :

```
mov     eax,uMsg
.if eax==WM_INITDIALOG
   invoke CheckRadioButton, hWin, RBN_ASM, RBN_C, RBN_ASM
.elseif eax==WM_COMMAND
   mov eax, wParam
   and eax, -1
   .if eax==BTN_CONV
      invoke GetDlgItemText, hWin, EDT_TEXT_IN, addr buf_in, sizeof buf_in
      push eax
      invoke IsDlgButtonChecked, hWin, RBN_ASM
      .if eax
         invoke HexDecode, addr buf_in, addr buf_tmp
         pop esi
         shr esi, 1
         invoke AsciiDump, addr buf_tmp, addr buf_out, esi

      .else
         invoke lstrcpy, addr buf_out, SADD("char array[] = {") ; copy array
         pop esi
         xor ecx, ecx
         mov edi, 16
         .while ecx < esi
            mov ax, word ptr buf_in[ecx]
            mov word ptr buf_out[edi], 'x0' ; concatenate hex prefix for c/c
            mov word ptr buf_out[edi+2], ax  ; concatenate the enetered byte
            mov word ptr buf_out[edi+4], ' ,'   ; concatenate separator
            add ecx, 2
            add edi, 6
         .endw
         mov word ptr buf_out[edi-2], ';}'   ; concatenate array end operato
         mov word ptr buf_out[edi], 0A0Dh ; (return)
         mov byte ptr buf_out[edi+2], 0 ; (null terminator)
```

و هذا مثال :

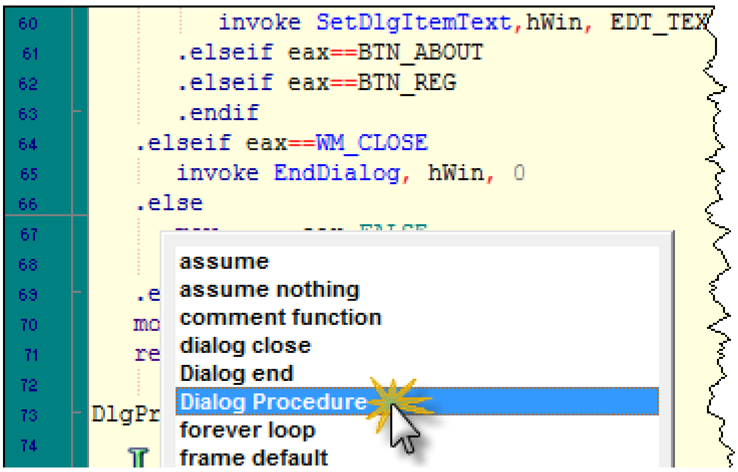


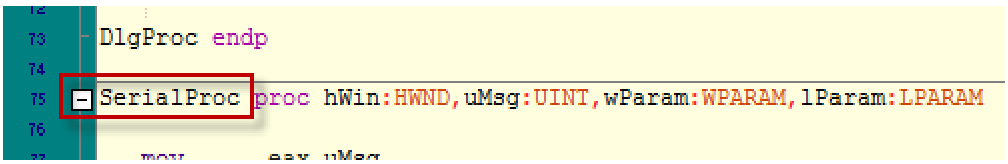
انتهت برمجة البريمج الصغير جداً (بالطبع لم يكن هدفنا أن نبرمج بقدر ما كان الهدف إيصال العديد من المعلومات حول إضافات RADASM).

1.3. إنشاء حماية للبرنامج

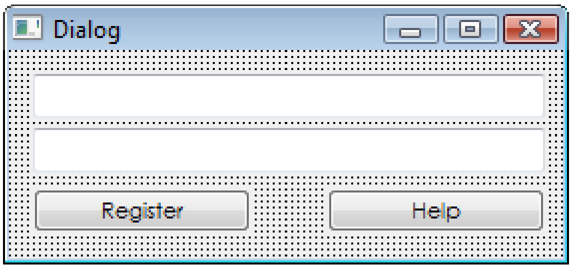
1.3.1. بطلب سريال واسم

ابتداء نستخدم ما تعلمنا CTRL+J ثم Dialog procedure :



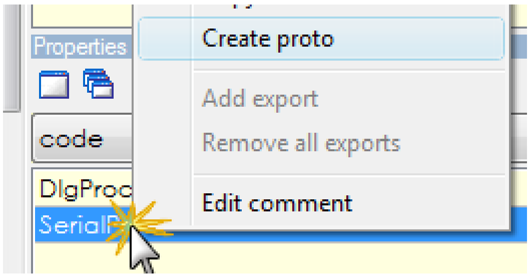


ثم استبدل اسم الدالة وأضف dialog بزرين ومربعي نص وأعد تسمية الـــ controls كما سبق إلى أسماء سهلة التذكر:




```
DLG_SERIAL              equ 102
EDT_NAME                equ 1001
EDT_SERIAL              equ 1002
BTN_REG_2               equ 1003
BTN_HELP                equ 1004
```

قبل استعمال الدالة SerialProc اعمل declaration لـــ proto الخاص بها، وذلك عن طريق الضغط بالزر الأيمن ثم Create proto بعد اختيار المكان المناسب :



و هذا هو الكود المضاف :

في الدالة DlgProc :

```
.elseif eax==BTN_REG
    invoke DialogBoxParam,hInstance,DLG_SERIAL,NULL,addr SerialProc,NULL
.endif
```

و :

```
mov    eax,uMsg
.if    eax==WM_INITDIALOG
   invoke   CheckRadioButton, hWin, RBN_ASM, RBN_C, RBN_ASM
   invoke GetPrivateProfileString,SADD("RegInfo"), \
                                  SADD("NAME"),\
                                  0, addr buf_in, \
                                  MAX_PATH, \
                                  SADD("reg.ini")
   mov dummy, eax
   invoke GetPrivateProfileString,SADD("RegInfo"), \
                                 SADD("SERIAL"), 0, \
                                 addr buf_out, \
                                 MAX_PATH, \
                                 SADD("reg.ini")
   invoke   IsSerialCorrectProc, addr buf_in, [dummy], addr buf_out
   .if !eax || dummy <= 6
      mov dummy, FALSE
   .else
      mov dummy, TRUE
```

و :

```
    .endif
.elseif  eax==WM_COMMAND
   mov   eax, wParam
   and   eax, -1
   .if   eax==BTN_CONV  && dummy
      invoke   GetDlgItemText, hWin, EDT_TEXT_IN, addr buf_in, sizeo
      push   eax
```

- في الدالة SerialProc :

ارجع إلى السورس

- في الدالة IsSerialCorrectProc

المسؤولة عن التحقق من السريال: اخترناها سهلة جداً؛ للتدرب عليها لاحقاً في هذا الدرس، ويمكنك دراستها إن أردت :

```
IsSerialCorrectProc  proc regname:DWORD, cname:DWORD, regserial:DWORD

LOCAL buf_1[MAX_PATH]:  BYTE

mov   edi, [regname]
mov   esi, [regserial]

xor   ecx, ecx

.while   [cname]
   movzx eax, byte   ptr [edi+ecx]
   shr   al, 1
   and   eax, 0ffh

@@:
   .if   eax < 13
      mov   al, byte ptr   array3[eax]
      mov   byte ptr buf_1[ecx], al
   .elseif  eax < 20 && eax >= 13
      mov   al, byte ptr   array2[eax]
      mov   byte ptr buf_1[ecx], al
   .elseif  eax < 31 && eax >= 20
      mov   al, byte ptr   array4[eax]
      mov   byte ptr buf_1[ecx], al
   .elseif  eax < 74 && eax >= 31
      mov   al, byte ptr   array1[eax]
      mov   byte ptr buf_1[ecx], al
   .else
      shr   al, 1
      and   eax, 0ffh
      jmp   @b
   .endif
   inc   ecx
   dec   [cname]
.endw

mov   byte ptr buf_1[ecx], 0
invoke   lstrcmp, addr buf_1, esi
.if   !eax
   mov   eax, TRUE
.else
   mov   eax, FALSE
.endif

ret

IsSerialCorrectProc  endp
```

و مخطط سير البرنامج العام كالتالي :

يقوم البرنامج بقراءة معلومات التسجيل من ملف reg.ini إذا كان موجوداً (في C:\windows) ويتحقق منها :

1 - إذا كانت صحيحة يجعل القيمة 1 في متغير.

2 - إذا لم تكن كذلك, يجعل قيمته 0.

إذا لم تكن قيمة المتغير 1 فإن البرنامج لن ينفذ التغيير إلى أي صيغة.

طبعاً الخوارزمية مبرمجة لتكون سهلة الكسر بجميع الأنواع الممكنة من الــ serial fishing إلى Keygen.

إن سهولة خوارزمية التسجيل أمر أردنا منه إشراك أكبر عدد من الأعضاء، لإكمال الدروس من جهة، ولتمكيننا من استخدام أكبر عدد من الأدوات التي نعتقد أن المبتدئين يجهلونها في عالم الهندسة العكسية من جهة أخرى.

1.3.2. باستخدام Packer/ Protector

نضغط البرنامج بواسطة برنامج الحماية السهل جدا (للتدريب فقط) وقد اخترنا برنامج الحماية القديم ACProtect 0.9g وذلك لسبب ستعرفونه فيما بعد.

البرنامج تجده في *Extra\Protection\Protector* \. مع نسختين من بريمجنا : واحدة محمية به والأخرى ليست محمية.

2. استعمال OllyDbg للتنقيح [خفايا OllyDbg]

2.1 اكتشاف الخطأ

نفترض أنه بعد نشرك لنسختك جاءك الرد التالي :

> ... ولكن عندما أدخل سطراً طويلاً، أكثر من 300 حرفــــــا فالبرنامج يهنج ويغلقه النظام. أنا أعمل على P4 .
>
> المرجو إصلاح الخطأ، وموافاتي بالنسخة المرقعة وشكراً.

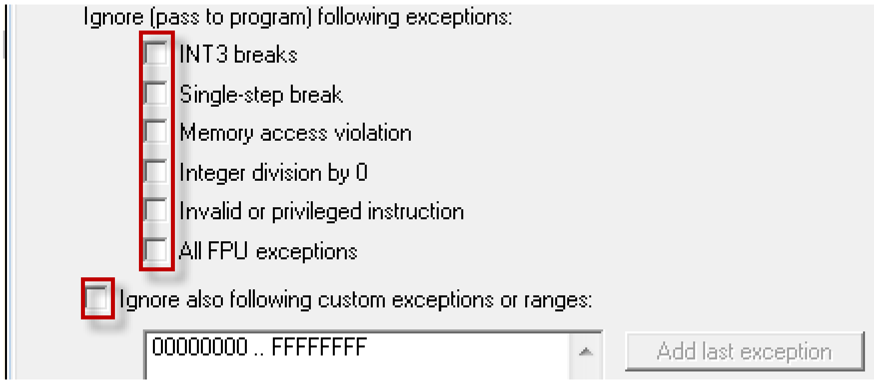
2.2. محاولة اكتشاف المكان المسؤول عن الخطأ

بمجرد قراءة الرسالة, يتبادر إلى ذهنك أن هنالك Buffer overflow هو المسؤول عن الخطأ؛ لأن المشكل يظهر فقط عندما يدخل المستعمل سلسلة محارف طويلة.

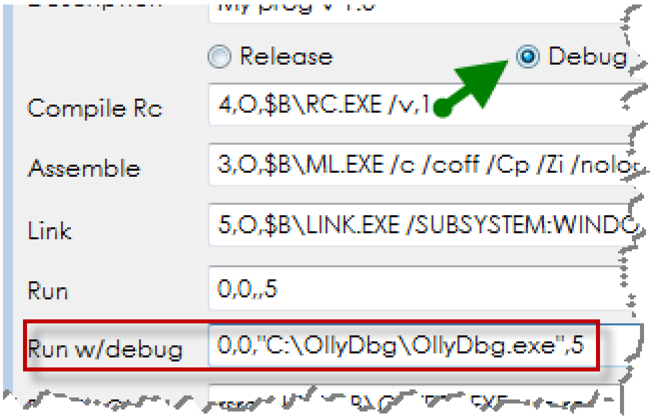
و لنفترض أن البرنامج كبير بما فيه الكفاية، وأن فيه استخدام مئات Buffers أثناء تحويل الــ DATA.

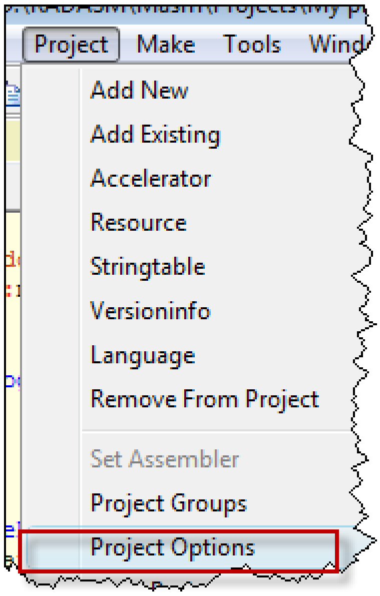
الآن سنحاول استخدام المنقح OllyDbg وعدد من المزايا الرائعة فيه.

افتح OllyDbg واضبطه هكذا (وهذا بالضبط هو الصواب أثناء التنقيح، للبحث عن أخطاء داخل البرامج) :



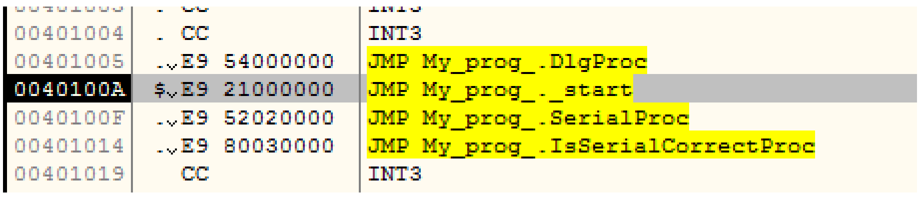
قبل أن تحمل الضحية اعمل compiling لنسخة Debug وذلك بتغيير الخيار التالي في RADASM في Project→Project Options :





اختر الخيار الأخضر وانتبه إلى المستطيل الأحمر ليكون المسار موجهاً إلى مكان OllyDbg (أو النسخة التي تستعمل منه) بحيث يسهل النداء عليه بـ CTRL+D

ثم اعمل Recompiling للمشروع واستدعِ OllyDbg بالضغط على CTRL+D، ستلاحظ أن OllyDbg تعرف على الأسماء التي استخدمتها من خلال تحميل الـــ symbols الموجودة في الداتا بايز التي يتم إنشاؤها أثناء الكومبايلينغ, وهذا يسهل التعامل مع الدوال والمتغيرات بشكل أحسن :



و هذا ليس كل شيء, اذهب في OllyDbg إلى View→Source files وسيفتح لك نافذة فيها جميع ملفات السورس التي استخدمتها في مشروعك، ويمكن اختيار أي واحد منها (في حالتنا لدينا فقط ملف واحد) :

Source files

Module	Source	Source path
My_prog_	MY PROG V 1.0.ASM	[redacted]My prog v 1.0\My prog v 1.0.asm

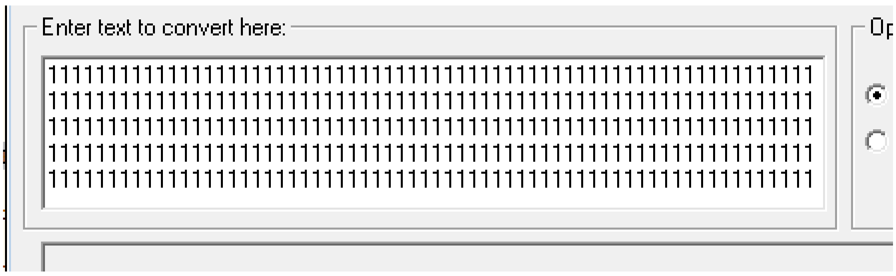
انقر عليه مرتين وستفتح ملف السورس :

```
>  42.                mov     dummy, TRUE
   43.            .endif
>  44.    .elseif eax==WM_COMMAND
>  45.            mov     eax, wParam
>  46.            and     eax, -1
>  47.            .if     eax==BTN_CONV  && dummy
>  48.                    invoke  GetDlgItemText, h
>  49.                    push    eax
>  50.                    invoke  IsDlgButtonChecke
>  51.                    .if     eax
>  52.                            invoke  HexDecode
>  53.                            pop     esi
>  54.                            shr     esi, 1
>  55.                            invoke  AsciiDump
   56.
>  57.                    .else
```

و بما أن الخطأ يحدث بعد الضغط على زر التحويل, فعليك بالذهاب إلى الوسالة WM_COMMAND وبالضبط إلى الحدث BTN_CONV وضع نقطة توقف هناك بــ F2 :

```
>  45.            mov     eax, wParam
>  46.            and     eax, -1
>  47.            .if     eax==BTN_CONV  && dummy
>  48.                    invoke  GetDlgItemText, hWin, EDT_TEXT
>  49.                    push    eax
>  50.                    invoke  IsDlgButtonChecked, hWin, RBN_
```

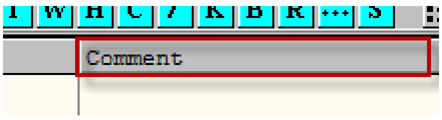
و أطلق البرنامج الآن ثم أدخل سلسلة محارف طويلة جداً :



ثم CONVERT وستتوقف في المكان المناسب :

```
0040112F  .v0F JNZ My_prog_.0040121B
00401135  . 83 CMP DWORD PTR DS:[dummy],0
0040113C  .v0F JE My_prog_.0040121B
00401142  . 68 PUSH 104
00401147  . 8D LEA EAX,DWORD PTR SS:[EBP-104]
0040114D  . 50 PUSH EAX
0040114E  . 68 PUSH 3E9
00401153  . FF PUSH DWORD PTR SS:[EBP+8]
00401156  . E8 CALL My_prog_._GetDlgItemTextA@16
0040115B  . 50 PUSH EAX
0040115C  . 68 PUSH 3F0
00401161  . FF PUSH DWORD PTR SS:[EBP+8]
00401164  . E8 CALL My_prog_._IsDlgButtonChecked@8
00401169  . 0B OR EAX,EAX
```

و لتتبع الأمور جيداً اضغط على الـــ bar الخاص بعمود التعليق لتغييره إلى عمود سورس :



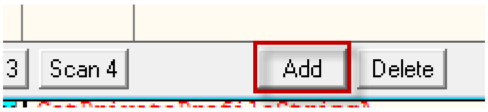
و ستلاحظ أن الكود والسورس أصبحا جنباً إلى جنب :

```
Disassembly                              Source
JNZ My_prog_.00401243
MOV EAX,DWORD PTR SS:[EBP+10]            mov     eax, wParam
AND EAX,FFFFFFFF                         and     eax, -1
CMP EAX,3EB                              .if     eax==BTN_CONV  && dummy
JNZ My_prog_.0040121B
CMP DWORD PTR DS:[dummy],0
JE My_prog_.0040121B
PUSH 104                                 invoke  GetDlgItemText, hWin, EDT_TEXT_IN, a
LEA EAX,DWORD PTR SS:[EBP-104]
PUSH EAX
PUSH 3E9
PUSH DWORD PTR SS:[EBP+8]
CALL My_prog_._GetDlgItemTextA@16
PUSH EAX                                 push    eax
PUSH 3F0                                 invoke  IsDlgButtonChecked, hWin, RBN_ASM
PUSH DWORD PTR SS:[EBP+8]
CALL My_prog_._IsDlgButtonChecked@8
OR EAX,EAX                               .if     eax
JE SHORT My_prog_.00401199
LEA EAX,DWORD PTR SS:[EBP-30C]           invoke  HexDecode, addr buf_in, addr buf_tmp
PUSH EAX
LEA EAX,DWORD PTR SS:[EBP-104]
PUSH EAX
CALL My_prog_._HexDecode@8
POP ESI                                  pop     esi
SHR ESI,1                                shr     esi, 1
PUSH ESI                                 invoke  AsciiDump, addr buf_tmp, addr buf_ou
```

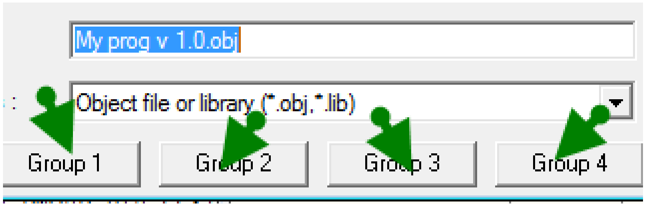
بالتتبع قليلاً في هذه الحالة ستعرف أن الخطأ في صغر الـــ Buffer الذي يتلقى سلسلة المحارف المحولة ولتصحيح الأمر نترك لك ذلك.

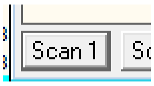
هناك طريقة أخرى لتحميل الـــ Symbols وذلك كالتالي :

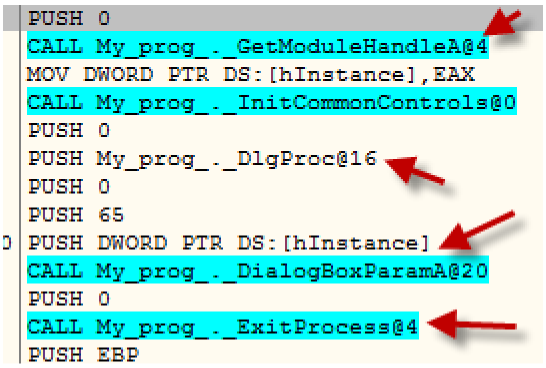
حمل نسخة Release من البريمج، اضغط CTRL+O ثم :



ثم أبحر إلى مكان وجود ملفات الــ Obj أو Lib الخاصة ببرنامجك وبالمكتبات التي أضفتها إليه، ثم اختر ما تريد، ثم اضغط أحد الأزرار الأربعة لترتيب الملفات إلى أربع مجموعات :



و بعد ذلك  لتشاهد الأسماء تظهر أمامك :



```
PUSH 0
CALL My_prog_._GetModuleHandleA@4
MOV DWORD PTR DS:[hInstance],EAX
CALL My_prog_._InitCommonControls@0
PUSH 0
PUSH My_prog_._DlgProc@16
PUSH 0
PUSH 65
PUSH DWORD PTR DS:[hInstance]
CALL My_prog_._DialogBoxParamA@20
PUSH 0
CALL My_prog_._ExitProcess@4
PUSH EBP
```

ملاحظات :

1 - يمكنك فعل نفس الشيء للمكتبة masm32.lib مثلاً، وكذلك المكتبة cryptohash.lib لاستيراد جميع أسمائهما.

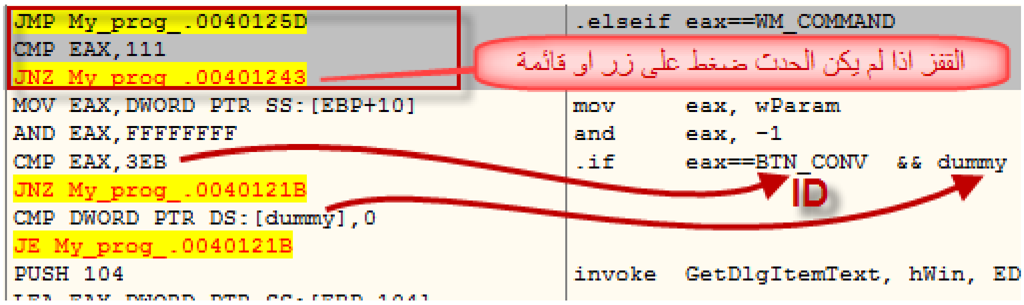
2 - خاصية تحميل السورس خاصة فقط بـــ COFF standard ولا يمكن تحميل ملفات الدلفي مباشرة مثلاً إلا بعد عمل compiling لـــ obj من خلال خصائص المشروع واختيار Produce obj c/c++ files وبعد ذلك يمكن استخدامها.

3 - و هذه الخاصية الأخيرة مهمة جداً؛ فمثلاً بعد التعرف على مكتبة هاشات ما في برنامج ما يمكنك تحميلها وعمل كومبايلينغ لها واستيراد الأسماء منها، مما يسهل الكثير من الأمور في التعرف على الدوال والمتغيرات دون الحاجة إلى إهدار الوقت في تتبع كل دالة خطوة بخطوة.

4 - عليك بدراسة ومقارنة ما تكتبه بالأسمبلي وما يتم إنتاجه كملف تنفيذي، ليسهل عليك فهم البرامج أثناء تنقيحها كهذه الأمثلة :



...



3. كسر البرنامج

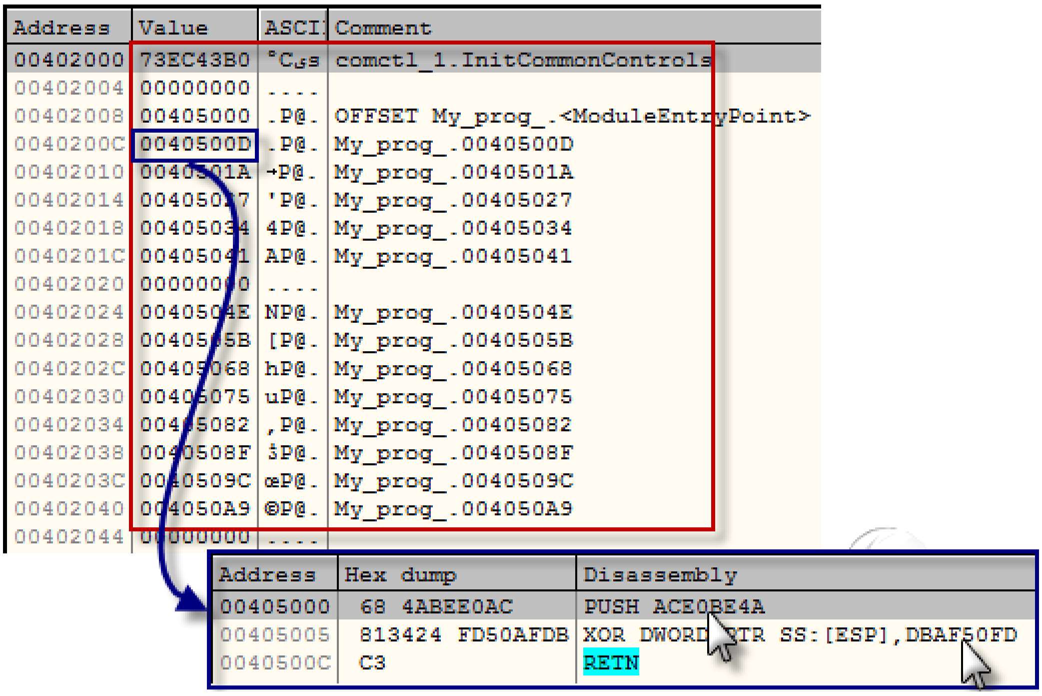
3.1 فك الضغط

لم يعد فك ACProtect مشكلة حتى عند المبتدئين, هذه نقطة نتفق معك فيها, لكن لا تغادر, ما رأيك إن قمت بكسره بطريقة احترافية, تمهد لك الطريق أمام الحمايات العتيدة؟

ما رأيك في برمجة plugin لـــ Imprec يقوم بإصلاح الـــ IAT (أجل, نعرف أنه من الممكن أن تعكس بعض القفزات وتكون الـــ IAT كاملة, لكن نريدك أن تتعلم الجديد).

حمل البرنامج المحمي إلى نسخة مموهة من OllyDbg كنسخة الأخ مراد من فريقنا. وأطلق البرنامج, ثم تتبعه وستجد بعض المقالب كـــ IsDebuggerPresent و CreateToolhelp32Snapshot ومن السهل تجاوزها.

لذا فلنركز على الـــ IAT والتي يقوم البرنامج بعمل xor للقيمة الصحيحة للـــ API مع قيمة عشوائية :



الـــ Plugin ستكون مهمته سهلة و ستتضمن الخطوات التالية:

1) استرجاع القيمة الأولى.

2) استرجاع القيمة الثانية.

3) عمل xor بين القيمتين.

4) إرجاع القيمة الصحيحة إلى Imprec ليقم بما يجب عليه.

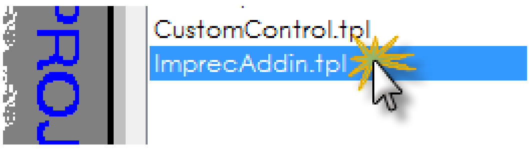
أما البرمجة فسهلة وموجودة في ملف المساعدة التابع لـــ Imprec المسمى Plugin.txt وتجده مع بعض الأمثلة.

المهم, استخدم Template المرفق مع الدرس (.\ImprecPlugin\Template\ImprecAddin.tpl).

بعد أن تستخرجها إلى %RADASM%\masm\templates قم بإنشاء مشروع جديد :



ثم يليها



و لم يبق الآن إلا أن تترجم الخطوات التي تكلمنا عنها سابقاً إلى كود فتصبح الدالة Trace على الشكل (إذا أردت استعمال Template مستقبلاً, فما عليك إلا أن تضيف كود التحقق بين طرفي المعقوفة وترك الكل كما هو) :

```
Trace proc hFileMap:DWORD, dwSizeMap:DWORD, dwTimeOut:DWORD, dwToTrace:DWORD,

    LOCAL ptrMap:DWORD

    invoke MapViewOfFile, hFileMap, FILE_MAP_READ or FILE_MAP_WRITE, 0, 0, 0
    .if eax
        mov ptrMap, eax
        .if dwSizeMap >= 4
            invoke IsBadReadPtr, [dwToTrace], 4
            .if !eax
                ; Enter your checking conditions here
                mov edi, [dwToTrace]
                mov esi, dword ptr [edi+5]
                and esi, 0FFFFFFh
                .if (byte ptr [edi] == 68h && esi == 243481h)
                    mov eax, dword ptr [edi+1]
                    xor eax, dword ptr [edi+8]
                    mov ebx, ptrMap
                    mov [ebx], eax
                    push IMP_SUCCESS
                .endif
            .else
                push IMP_BAD_POINTER
            .endif
        .else
            push IMP_INVALID_MAP_SIZE
        .endif
    .else
        mov eax, IMP_CANT_MAP_VIEW
        ret
    .endif

    invoke UnmapViewOfFile, ptrMap
    invoke CloseHandle, hFileMap
    pop eax
    ret
```

يوضع الكود في مكان المعقوفة

وإليك الشرح :

1. الخطوة الأولى :

الكود	`mov edi, [dwToTrace]`	Edi الآن مؤشر نحو الـ IAT المنحرفة
مقابله في Imprec	? FThunk:00002008 NbFunc:6 (decimal:6) valid:NO rva:00002008 ptr:00405000	
مقابله في OllyDbg	Address / Hex dump / Disassembly 00405000 68 6CBA8924 PUSH 2489BA6C 00405005 813424 DB54C653 XOR DWORD PTR SS:[ESP], 53C654DB 0040500C C3 RET	

2. الخطوة الثانية :

الكود	`mov esi, dword ptr [edi+5]` `and esi, 0FFFFFFh`	عزل التعليمة الثانية
مقابله في OllyDbg	Address | Hex dump | Disassembly 00405000 | 68 6CBA8924 | PUSH 2489BA6C 00405005 | 813424 DB54C653 | XOR DWORD PTR SS:[ESP], 53C654DB 0040500C | C3 | RET	

3. الخطوة ألثالثة :

الكود	`.if (byte ptr [edi] == 68h && esi == 243481h)`	التأكد من التعليمات
الشرح	هل التعليمة الأولى Push والثانية Xor [esp] ؟	

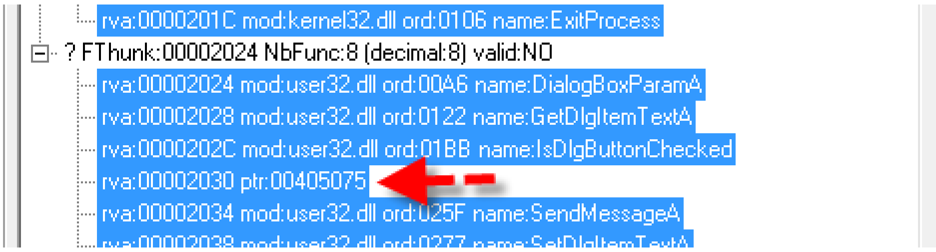
4. الخطوة الرابعة :

الكود	`mov eax, dword ptr [edi+1]` `xor eax, dword ptr [edi+8]` `mov ebx, ptrMap` `mov [ebx], eax` `push IMP_SUCCESS`	عمل الـ XOR
مقابله في OllyDbg	Address | Hex dump | Disassembly 00405000 | 68 6CBA8924 | PUSH 2489BA6C xor 00405005 | 813424 DB54C653 | XOR DWORD PTR SS:[ESP], 53C654DB 0040500C | C3 | RET	

5. الخطوة الخامسة :

الكود	`mov eax, dword ptr [edi+1]` `xor eax, dword ptr [edi+8]` `mov ebx, ptrMap` `mov [ebx], eax` `push IMP_SUCCESS`	حساب القيمة الصحيحة و ارجاع القيمة 200 (التي *تعني تمام الامر بصورة صحيحة*)
الشرح	احسب ناتج الـ xor واحفظه بحيث يصبح في الكلمة المزدوجة الأولى من Map، ومرر القيمة IMP_SUCCESS أي (200) وتعني نجاح العملية.	

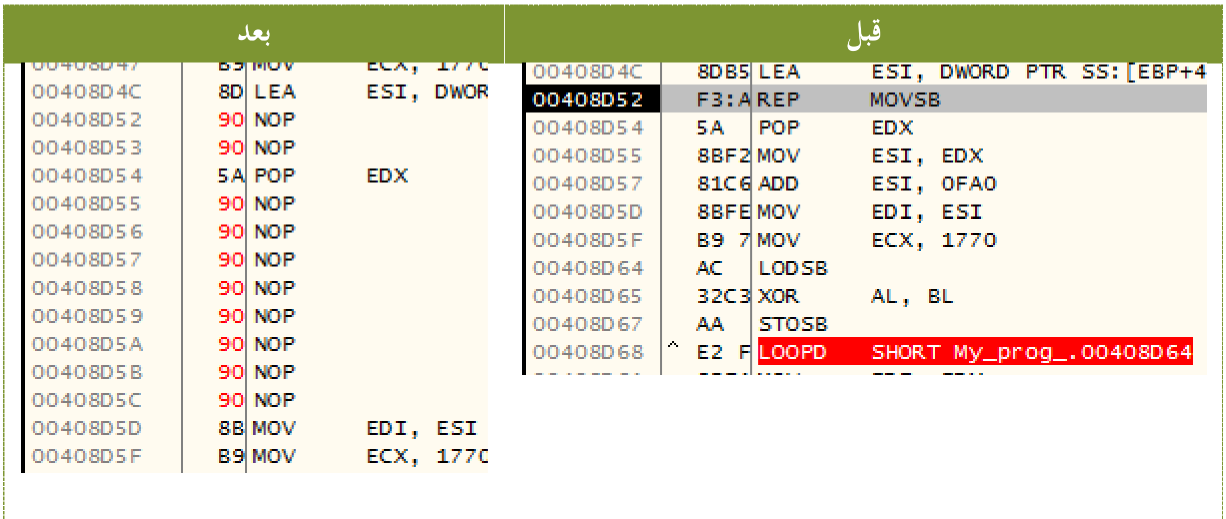
و الناتج بعد نسخ الــ plugin في المسار المناسب لــ Imprec :

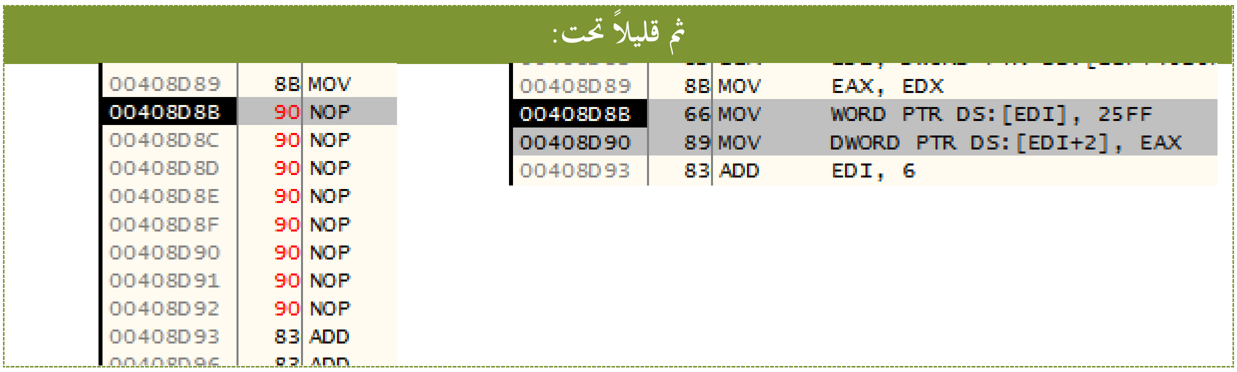


وإذا تتبعت البرنامج بــ OllyDbg ستجد أن الــ API الناقص هو MessageBoxA ويمكن أن تضيفه بالنقر مرتين على السطر ثم اختيار الاسم المناسب.

وإليك بعض الــ Hints لإتمام الــ unpacking :

1. ضع نقطة على GlobalAlloc وتوقف في المرة الثالثة عليها (F9 ثلاث مرات).
2. عند العودة إلى الكود سيظهر الكود : غيّر كما في الصورة :





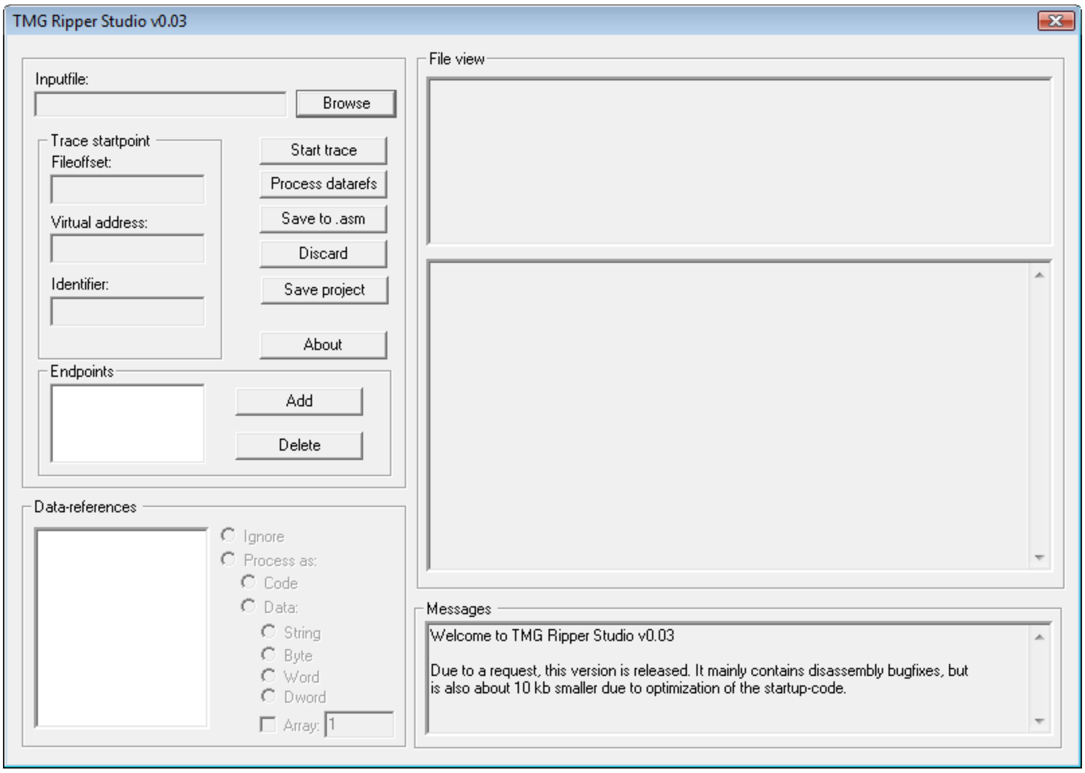
3. F2 على text. Section ثم Dump ثم Reb IAT ثم RUN ثم مبلوك عليك.

3.2 كسر الحماية بكيجن

سنتطرق اليوم إلى أحد المعدات الهامة، والتي تختصر عليك الوقت(طبعاً بالإمكان استعمال Coderipper أو IDA مباشرة, لكن تعلم الشيء الجديد أمر جميل).

3.2.1 TMG Ripper Studio [كيجن في دقيقة واحدة]

الأداة الأولى ستكون TMG Ripper Studio الرائع في مثل هذا المثال البسيط، وسنفترض أنك عرفت كيف تحدد مكان الخوارزمية ولا تحتاج إلا لنسخها. افتح البرنامج (KeyGenning\TMG\Ripper\.) :



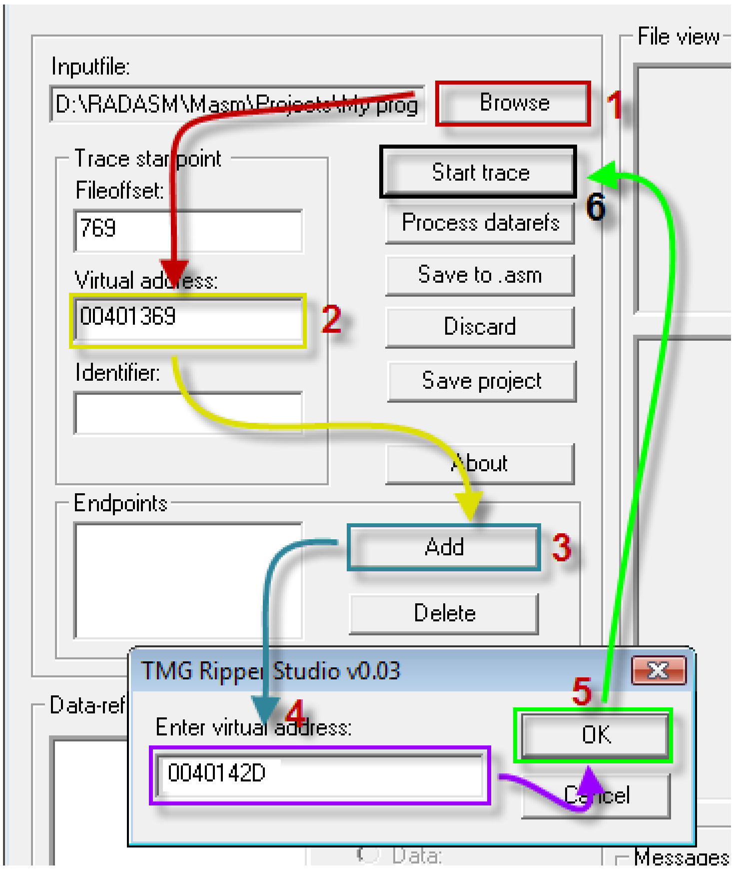
الآن ما عليك إلا أن تملأ بعض المعلومات والبرنامج يكمل ا لباقي (ملاحظة هامة: لقد عدلنا في آخر لحظة على خوارزمية التسجيل لنظهر لك أهمية وقوة البرنامج، وأين تكمن؟ والهدف المستعمل ستجده في Extra\Keygenning\TMG\Target.).

هذه أهم المعلومات :

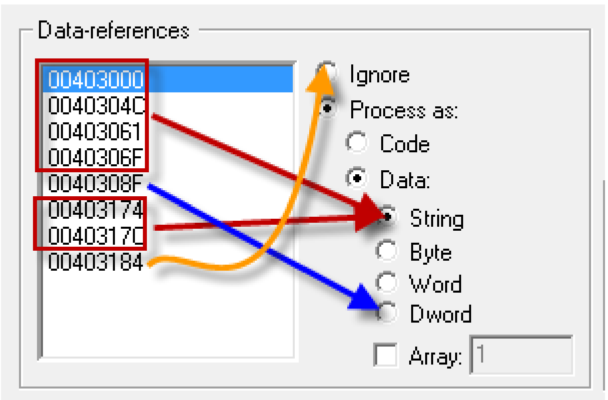
1 - بداية الخوارزمية في: 00401369

2 - نهايتها في 0040142D قبيل المقارنة.

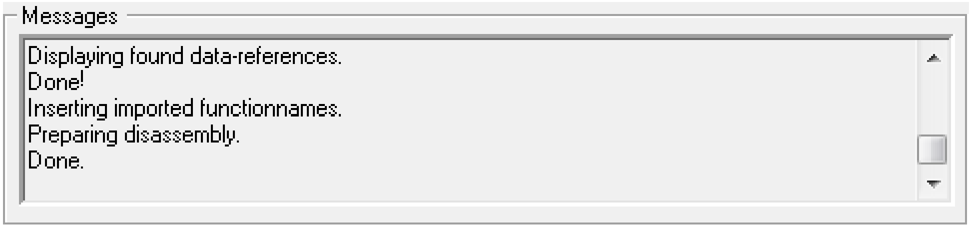
دع OllyDbg مفتوحاً، وإليك المراحل بالتتابع :



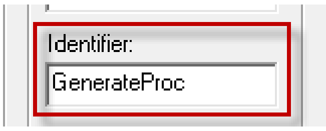
وإليك التفاصيل (طبيعي أن يسبق هذا دراسة للخوارزمية، ويمكنك التتبع أيضاً في الخانات على اليمين في TMGRS للمساعدة) :



بعد تحديد الأنواع الصحيحة اضغط: Start trace ثم Process datarefs وبعد ذلك يظهر لك :



وتختفي المعلومات في data references يمكنك إدخال اسم للدالة في



و الآن لم يبق إلا الضغط على Save to .asm لحفظ المشروع وتعديل بعض السطور ويصبح الكيجن جاهزاً:

افتحه بمحرر نصوص، ثم اذهب إلى السطور الأخيرة، وأعد تسمية الــAPI في كل النص ثم احذف السطور :

```
    LEAVE
    RETN                 00008h
LOC_004014EE:    ;Ref: 0040147D, 004014CA
    JMP                  lstrcatA
LOC_004014FA:    ;Ref: 00401461, 0040146E
    JMP                  lstrcpyA
LOC_00401500:    ;Ref: 004014FA
    JMP                  wsprintfA
```

يجب أيضاً إضافة سطرين في أول الكود :

```
OPTION PROLOGUE:NONE
OPTION EPILOGUE:NONE

GenerateProc:
    PUSH                 EBP
```

و سطرين في آخره :

```
    LEAVE
    RETN                 00008h

OPTION PROLOGUE:PROLOGUEDEF
OPTION EPILOGUE:EPILOGUEDEF
```

و فائدتهما : إجبار الكمبايلر على عدم إضافة أي prologue ولا epilogue للدوال لأنها تملك واحدة.

كذلك أضف كلمة proc للدوال المتعرف عليها؛ للتمكن من المناداة عليها بسهولة:

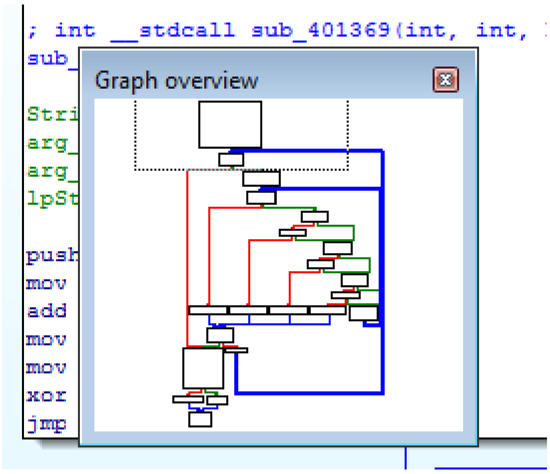
```
GenerateProc proc:
    PUSH                 EBP
```

ارجع إلى السورس للمقارنة.

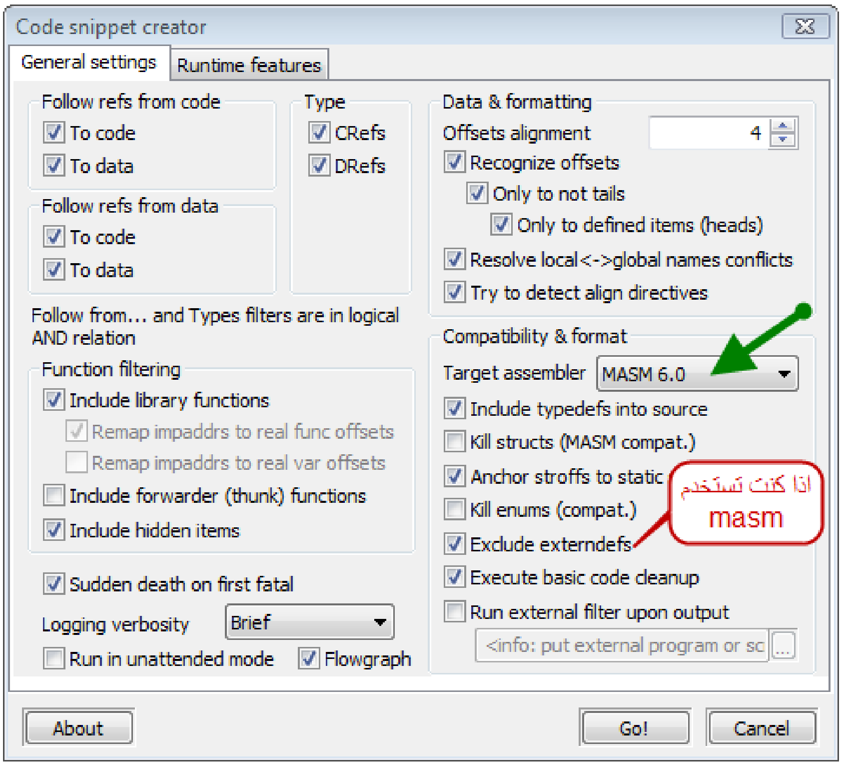
3.2.2. باستعمال IDA+PLUGIN [كيجين في 30 ثانية]

البلاغين موجود في \Keygenning\IDA\Plugin. انسخه في %IDA%\plugins .

ما عليك إلا أن تتوجه إلى الخوارزمية:



و من ثم اذهب إلى Edit→Code snipet creator أو اضغط ALT+F10 :



وعنده ا سيتولد الكيجن.

ما عليك إلا تغيير 3 أو 4 أسطر كما في السابق والنتيجة فورية.

السورس مرفق في Keygenning\IDA\Keygen\.

هذا الدرس لم يكن من تحدي الخوارزميات اللامعكوسة أو تحدي برامج حماية صعبة، بل فقط لإيصال أكبر عدد من المعلوامت التي نظن أنها مغيبة عن المبتدئين، والتي تشكل في نظرنا قواعداً هامة يجب على المهندس العكسي أن يتقنها، وعلى المبتدئ أن يصل إليها يوماً ما. ولهذا السبب لم نعر الاهتمام الواجب للبرنامج (إذا صح تسميته كذلك) من حيث الاعتناء بالأخطاء التي قد تحدث، لذا فاعذرونا.

التطفل للتمكن...

Malwares Hooking



I. المقدمة

تعتمد الفيروسات أو البرامج الضارة على عدة طرق لتنقل وتلويث النظام؛ منها من يعيش في Kernel land ومنها من يعيش في User land قوة Kernel land تكمن في الحصول على كل صلاحيات النظام على النظام، يعني التحكم الكامل في النظام، وهذا ما تعتمد عليه مضادات البرامج الضارة الحديثة، برامج التخريب والبرامج التي تنسخ نفسها بدون أي هدف وهي أبسط نوع من البرامج الضارة.



- Delphi7.
- VMWare صنع نظام وهمي للتجربة.
- AT4RE Faust Virus Remover لفحص الـ Process
- AT4RE Kernel Detective لإلغاء الهوك.



المرفقات في الرابط (اضغط على الصورة).

تم تقسيم الدرس إلى قسمين :

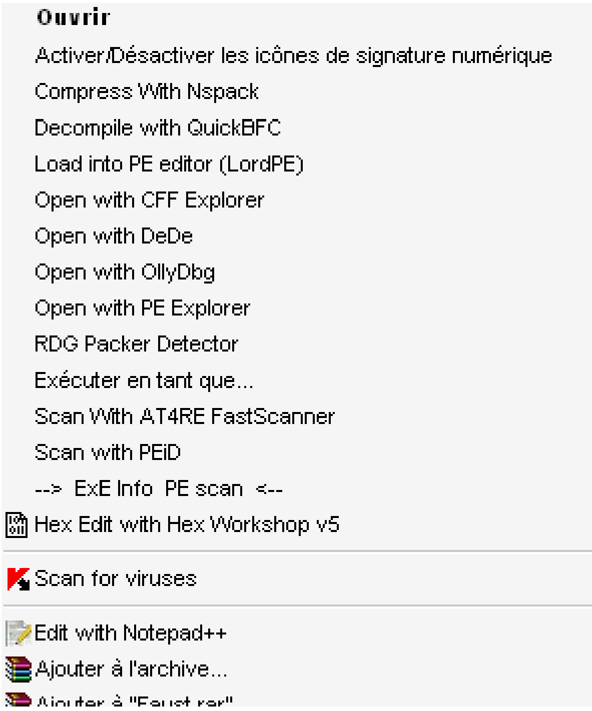
1 - عمل Hook لــ Explorer.exe ومنه التحكم في تشغيل أو إبطال خصائص معينة سنتطرق إليها إن شاء الله في هذا الدرس.

2 - عمل Hook لــ Winlogon.exe لجعل حذف الملف الضار أمراً صعبًا

2. Windows Explorer متصفح النظام

متصفح النظام Windows explorer يسمح بدمج إضافات للبرامج؛ مثل برامج الضغط وفك الملفات WinRar ومضادات الفيروسات، لتسهيل التعامل معها من طرف المستعمل، ويكفي فقط تسجيلها في الريجستري ليحملها المتصفح تلقائيا .

سماح النظام بتسجيل واستدعاء البرامج بهذه السهولة، جعل مبرمجي الفيروسات يستغلونها لتشغيل برامجهم الضارة والاعتماد الكامل على الــ Hooking لتخطي الجدران النارية وتحميل نسخ أحدث للفيروسات:



قائمة إضافات لبرامج غير ضارة يسمح بها النظم

سنحاول في هذا الدرس فهم آلية الــHooking وتطبيق مثالين عملين تمت برمجتهم بلغة دلفي مرفقين مع هذا الدرس

2.1 GUID - Globally Unique Identifier

هو رقم تسلسلي وحيد، يستعمل للتعريف بالبرامج في الريجستري، يعتمد عليه المبرمجون لتجنب الخلط بين القيم التي تكتب في الريجستري، وسنستعمله في عملية تسجيل ملف الهوك.

للمزيد من المعلومات تصفح الرابـــــــــــــــــط التالي :

للحصول على GUID يمكننا استعمال برامج التوليد الخاصة به، أو الاعتماد على دلفي لتوليد رقم

الطريقة :

في نافذة دلفي الخاصة بالكود اضغط على الأزرار التالية دون أن ترفع يدك: Ctrl + Shift + G

```
Unit1.pas
Unit1

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
['{0494E124-B95B-4A44-BD50-6B24CE4FE53D}']
['{88A772BB-2960-423C-9196-E7908177A07A}']
['{FA58A788-0787-4D75-BBDB-3E03ACDCC5E0}']
['{9FA15FE1-9C03-49B6-A337-ABC05AC74689}']
['{A637E865-EB3D-4606-81B2-AB1FB5A37402}']
['{02302569-4CE0-4BA9-88CD-DDCE87D62EE2}']
['{92558E54-A4B6-4E86-ADA7-84D2B1508DD3}']
['{1644952C-4B95-4082-B5B3-D3F9C85E4035}']
['{D20209ED-AA17-4970-90BA-0E3F07BCCFFA}']
['{9A8176FB-43E0-43C9-9CC3-97F9CEB4E825}']
['{76BEC85F-C88D-463B-A6E7-4D5DF4078EC4}']
['{EADC3A03-7FAB-4116-BBD3-3AFC738DE03A}']
end;
```

Ctrl + Shift + G لتوليد أرقام وحيدة لا متناهية

في مثالنا سنأخذ هذا الرقم : {E124-B95B-4A44-BD50-6B24CE4FE53D0494} ، وسنعمل على الملف المرفق مع نسخة دلفي 7 في مجلد Demos .

المسار : X:\Program Files\Borland\Delphi7\Demos\ActiveX\ShellExt

سنقوم ببعض التغييرات البسيطة لنتمكن من إبطال خصائص النظام التالية Copy – Move – Delete - .Rename

2.2. ملف الهوك – MalwareHook.pas

2.2.1. الجزء الخاص بـ Declaration

```
unit MalwareHook;

interface
```

```
uses
  Windows, ActiveX, ComObj, ShlObj;

type
  TCopyHook = class(TComObject, ICopyHook)
  public
    function CopyCallback(Wnd: HWND; wFunc, wFlags: UINT; pszSrcFile: PAnsiChar;
      dwSrcAttribs: DWORD; pszDestFile: PAnsiChar; dwDestAttribs: DWORD): UINT;
stdcall;
  end;

const
  Class_CopyHook: TGUID = '{0494E124-B95B-4A44-BD50-6B24CE4FE53D}';
implementation

uses ComServ, ShellAPI, SysUtils, Registry;
```

2.2.2. الجزء الخاص بدالة التعرف على العمليات الخاصة بالمجلدات

يمكن تغييرها بأي دالة أخرى حسب الحاجة :

```
function TCopyHook.CopyCallBack(Wnd: HWND; wFunc, wFlags: UINT; pszSrcFile:
PAnsiChar;
  dwSrcAttribs: DWORD; pszDestFile: PAnsiChar; dwDestAttribs: DWORD): UINT;
// This is the method which is called by the shell for folder operations
const
  ConfirmMessage = 'Sorry :) | This function is disabled'#13
    + 'System Hooking | Only for Educational use '#13
    + ''#13
    + 'Arab Team 4 Reverse Engineering'#13
    + 'STRELiTZIA | AT4RE'#13
    + 'www.at4re.com';
var
  Operation: string;
begin
  case wFunc of
    FO_COPY:
      Operation := 'copy';
    FO_DELETE:
      Operation := 'delete';
    FO_MOVE:
      Operation := 'move';
    FO_RENAME:
      Operation := 'rename';
  else
    Operation := '0';
  end;

  Result := MessageBox(Wnd, PChar(Format(ConfirmMessage, [Operation,
pszSrcFile])),
    'Malware Hooking sample',MB_OK);
end;
```

2.2.3 الجزء الخاص بالتسجيل في النظام

يحتوي على الأوامر التي سوف تنفذ من طرف برنامجregsvr32 لتسجيل أو إلغاء تسجيل ملف الهوك :

```
type
  TCopyHookFactory = class(TComObjectFactory)
  public
    procedure UpdateRegistry(Register: Boolean); override;
  end;

procedure TCopyHookFactory.UpdateRegistry(Register: Boolean);
var
  ClassID: string;
begin
  if Register then begin
    inherited UpdateRegistry(Register);

    ClassID := GUIDToString(Class_CopyHook);
    CreateRegKey('Directory\shellex\CopyHookHandlers\MalwareHookSample', '',
ClassID);

    if (Win32Platform = VER_PLATFORM_WIN32_NT) then
      with TRegistry.Create do
      try
        RootKey := HKEY_LOCAL_MACHINE;
        OpenKey('SOFTWARE\Microsoft\Windows\CurrentVersion\Shell Extensions',
True);
        OpenKey('Approved', True);
        WriteString(ClassID, 'Malware Hooking Example');

      finally
        Free;
      end;
  end
  else begin
    DeleteRegKey('Directory\shellex\CopyHookHandlers\MalwareHookSample');

    inherited UpdateRegistry(Register);
  end;
end;

initialization
  TCopyHookFactory.Create(ComServer, TCopyHook, Class_CopyHook, '',
    'Malware Hook Example', ciMultiInstance, tmApartment);
end.
```



2.3 ملف التثبيت SETUP.DPR

2.3.1 الجزء الخاص بـ Declaration

```
program Setup;

uses SysUtils, Windows;
```

2.3.2 الجزء الخاص بدالة تمرير بارومترات التسجيل من ملف الهوك إلى برنامج التسجيل REGSVR32

```
function RegisterServer(const aDllFileName: string; aRegister: Boolean): Boolean;
type
  TRegProc = function: HResult;
  stdcall;
const
  cRegFuncNameArr: array[Boolean] of PChar =
  ('DllUnregisterServer', 'DllRegisterServer');
var
  vLibHandle: THandle;
  vRegProc: TRegProc;
begin
  Result := False;
  vLibHandle := LoadLibrary(PChar(aDllFileName));
  if vLibHandle = 0 then Exit;
  @vRegProc := GetProcAddress(vLibHandle, cRegFuncNameArr[aRegister]);
  if @vRegProc <> nil then
    Result := vRegProc = S_OK;
  FreeLibrary(vLibHandle);
end;
```

2.3.3 الجزء الخاص بالمستعمل في تفعيل أو الغاء التسجيل

```
begin
  case MessageBox(0, 'Click YES - To Install HOOK - Restart your system'#13'Click
NO - To Uninstall HOOK - Restart your system', ' Malware System hooking sample',
MB_YESNOCANCEL + MB_ICONQUESTION) of
```

```
    ID_YES: if RegisterServer('MalwareHookTest.dll', True) then MessageBox(0,
'HOOK Installed', 'Sucess', MB_OK + MB_ICONINFORMATION);
    ID_NO: if RegisterServer('MalwareHookTest.dll', False) then MessageBox(0,
'HOOK Unistalled', 'Sucess', MB_OK + MB_ICONINFORMATION);
  end;
end.
```

ملاحظة : لو كان فيروساً حقيقياً، فان المبرمج لن يعطيك الخيار لإلغاء التسجيل، والـــ Exports التابعة للمثال المرفق لن تكون بهذا الشكل:

1- DllGetClassObject.
2- DllCanUnloadNow.
3- DllRegisterServer.
4- DllUnregisterServer.

تكون هكذا لتفادي إلغاء تسجلي الهوك :

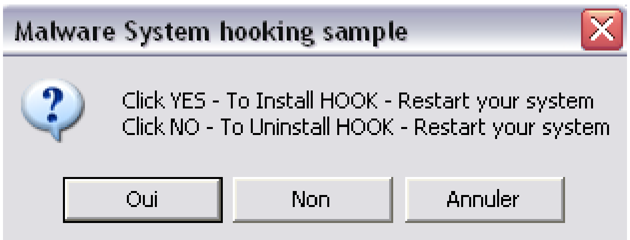
1- DllGetClassObject.
2- DllRegisterServer.

3. الجزء التطبيقي

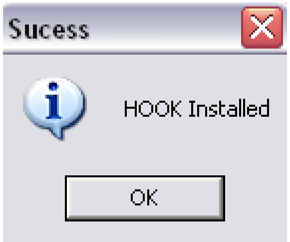
نشغل VMWare وننقل إليها الملفين التاليين: MalwareHookTest.dll وSETUP.exe :



نشغل ملف تسجيل الهوك SETUP.exe :



نضغط الزر موافق :



تظهر لنا رسالة نجاح العملية.

> لو كان الفيروس حقيقياً لما ظهرت هذه الرسائل كما ذكرنا آنفاً.

3.1 تفحص الريجستري

نلاحظ إنشاء قيم جديدة في الريجستري بعد إتمام عملية تثبيت الهوك :

تسجيل ملف الهوك بمعرف GUID :

```
[HKEY_CLASSES_ROOT\CLSID\{0494E124-B95B-4A44-BD50-6B24CE4FE53D}]
@="Malware Hook Example"

[HKEY_CLASSES_ROOT\CLSID\{0494E124-B95B-4A44-BD50-6B24CE4FE53D}\InprocServer32]
@="C:\\DOCUME~1\\UTILIS~1\\Bureau\\MALWAR~1.DLL"
"ThreadingModel"="Apartment"
```

إضافة المعرف الذي أصبح يعمل عمل Pointer إلى Classe Id – CLSID الخاصة بملف الهوك :

```
[HKEY_CLASSES_ROOT\Directory\shellex\CopyHookHandlers\MalwareHookSample]
@="{0494E124-B95B-4A44-BD50-6B24CE4FE53D}"
```

تمت إضافة معرف GUID أيضا إلى قائمة Approved :

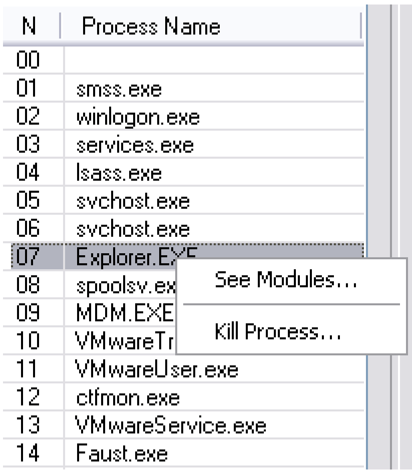
```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Shell
Extensions\Approved]
"{8DD448E6-C188-4aed-AF92-44956194EB1F}"="Windows Media Player Play as Playlist
Context Menu Handler"
"{CE3FB1D1-02AE-4a5f-A6E9-D9F1B4073E6C}"="Windows Media Player Burn Audio CD
Context Menu Handler"
"{F1B9284F-E9DC-4e68-9D7E-42362A59F0FD}"="Windows Media Player Add to Playlist
Context Menu Handler"
"{BDEADF00-C265-11D0-BCED-00A0C90AB50F}"="Dossiers Web"
"{00020D75-0000-0000-C000-000000000046}"="Microsoft Office Outlook Desktop Icon
Handler"
"{0006F045-0000-0000-C000-000000000046}"="Microsoft Office Outlook Custom Icon
Handler"
```

```
"{42042206-2D85-11D3-8CFF-005004838597}"="Microsoft Office HTML Icon Handler"
"{B41DB860-8EE4-11D2-9906-E49FADC173CA}"="WinRAR shell extension"
"{0494E124-B95B-4A44-BD50-6B24CE4FE53D}"="Malware Hooking Example"
…..
…..
…..
…..
```

تمت عملية تسجيل الهوك بنجاح في الريجستري، و لم يبق إلا التأكد من فحص الــ Modules الخاصة بمتصفح النظام Windows Explorer.

3.2 تفحص الـ Modules

نقوم بتشغيل برنامج AT4RE Faust Virus Remover ثم نذهب إلى Tools ثم Kill process :



نختار Explorer.exe ثم من قائمة الزر الأيمن نضغط على See Modules :

```
Module Number : 67 ===============
Name          : C:\DOCUME~1\UTILIS~1\Bureau\MALWAR~1.DLL
Size          : 136 Ko

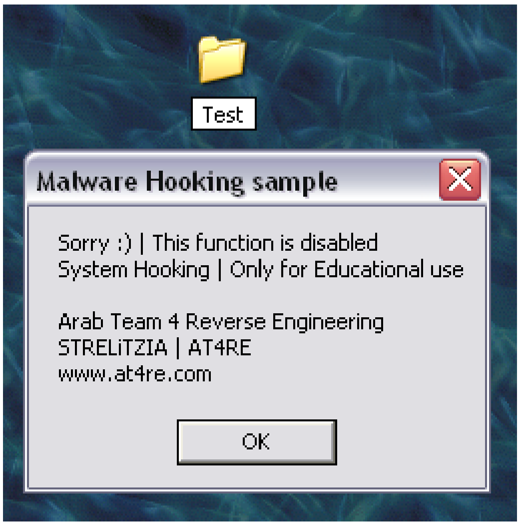
Module Number : 68 ===============
Name          : C:\WINDOWS\system32\mydocs.dll
```

نلاحظ أن ملف الهوك تم تحميله بنجاح، و لم يبق لنا إلا أن نجرب الملف

3.3 تجربة عمل ملف الهوك

ذكرنا آنفا أن ملف الهوك يعطل الخصائص التالية:

Copy – Move – Delete - Rename



عند محاولة إعادة تسمية المجلد تظهر لنا هذه الرسالة وهي رسالة ملف الهوك:



ثم تليها رسالة الخطأ الخاصة بالنظام وتبطل عملية إعادة التسمية.

نفس النتيجة مع تجربة الخصائص الأخرى. الآن تأكدنا أن الهوك يعمل بنجاح

3.4. عمل هوك لملف Winlogon.exe لحماية ملف الهوك من الحذف

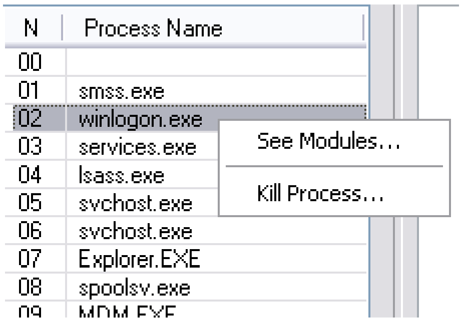
لحماية متوسطة لملف الهوك من الحذف، سنستعمل طريقة تعتمد على هوك أيضاً لملف حساس، وهو Winlogon.exe لأن كل الملفات المحملة معه لا تقبل الحذف.

نضيف مفتاحاً جديداً في الريجستري كالتالي :

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Winlogon\Notify\MalwareHookTest]
"Asynchronous"=dword:00000001
"DllName"="C:\\Documents and Settings\\Utilisateur\\Bureau\\MalwareHookTest.dll"
"Impersonate"=dword:00000000
```

```
"Logon"="o"
"Logoff"="f"
```

يجب تغيير مسار ملف الهوك DllName حسب الحاجة وإعادة تشغيل النظام.



نختار Winlogon.exe ثم من قائمة الزر الأيمن نضغط على See Modules :

```
C:\WINDOWS\system32\SAMLIB.dll
76 Ko

51 ===============
C:\Documents and Settings\Utilisateur\Bureau\MalwareHookTest.dll
136 Ko

52 ===============
C:\WINDOWS\system32\oleaut32.dll
560 Ko

53 ===============
```

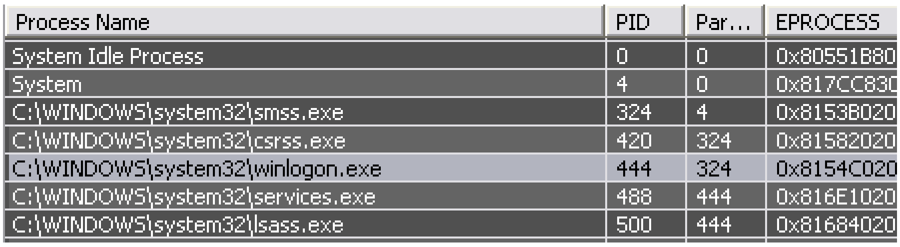
نلاحظ أن ملف الهوك تم تحميله بنجاح، وأصبح حذفه أمراً صعباً

> ملاحظة : يستعمل عادة هوك للملف Winlogon.exe لإقلاع الفيروسات.

3.5. إلغاء الهوك

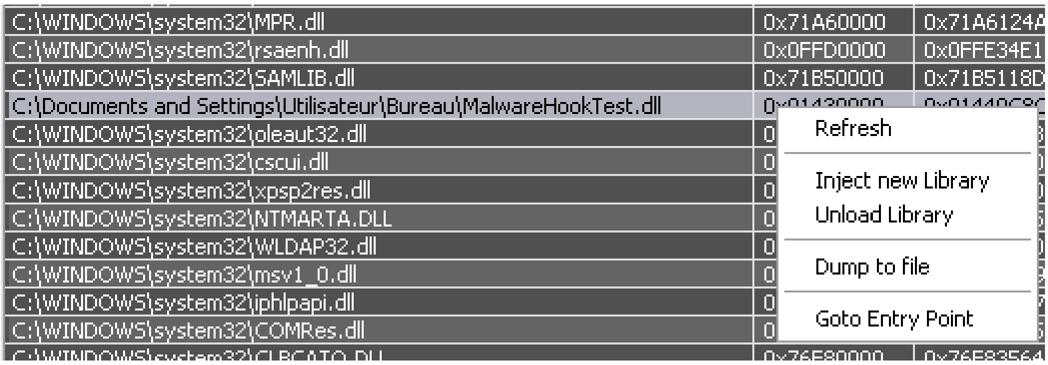
في هذا المثال يمكنك أخي القارئ أن تلغيَ الهوك بنفس البرنامج الذي تمت به عمليه التثبيت، ولكن ليس في حالة كونه فيروساً حقيقياً؛ لذا سنستعمل خاصية Unload لبرنامج الفريق Kernel Detective .

3.6. تنظيف Winlogon.exe



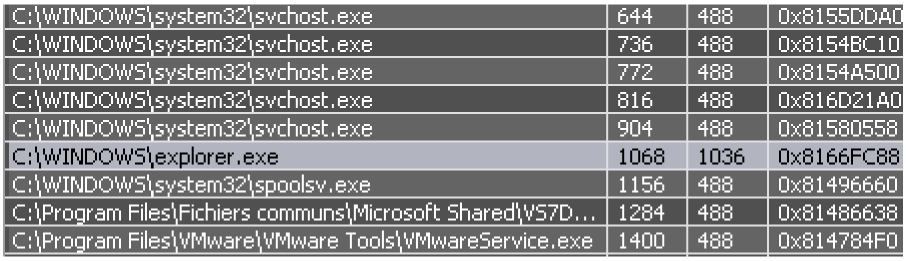
Process Name	PID	Par...	EPROCESS
System Idle Process	0	0	0x80551B80
System	4	0	0x817CC830
C:\WINDOWS\system32\smss.exe	324	4	0x8153B020
C:\WINDOWS\system32\csrss.exe	420	324	0x81582020
C:\WINDOWS\system32\winlogon.exe	444	324	0x8154C020
C:\WINDOWS\system32\services.exe	488	444	0x816E1020
C:\WINDOWS\system32\lsass.exe	500	444	0x81684020

نختار من قائمة الـــ Process ملف Winlogon.exe ثم نذهب إلى تبويب Modules :



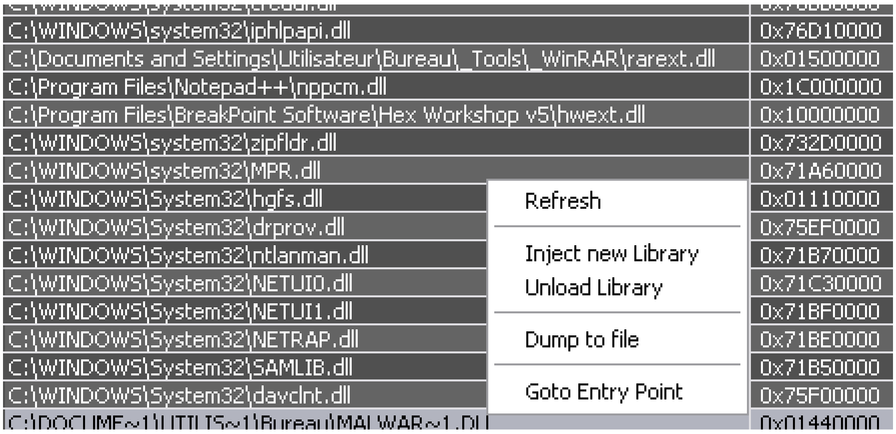
نختار ملف الهوك ثم Unload Library .

3.7. تنظيف Explorer.exe



C:\WINDOWS\system32\svchost.exe	644	488	0x8155DDA0
C:\WINDOWS\system32\svchost.exe	736	488	0x8154BC10
C:\WINDOWS\system32\svchost.exe	772	488	0x8154A500
C:\WINDOWS\system32\svchost.exe	816	488	0x816D21A0
C:\WINDOWS\system32\svchost.exe	904	488	0x81580558
C:\WINDOWS\explorer.exe	1068	1036	0x8166FC88
C:\WINDOWS\system32\spoolsv.exe	1156	488	0x81496660
C:\Program Files\Fichiers communs\Microsoft Shared\VS7D...	1284	488	0x81486638
C:\Program Files\VMware\VMware Tools\VMwareService.exe	1400	488	0x814784F0

نختار من قائمة الـــ Process ملف Explorer.exe ثم نذهب إلى تبويب Modules :



نختار ملف الهوك ثم Unload Library .

لم يبقَ لنا الآن إلا أن نحذف القيم التي أضيفت إلى الريجستري، وحذف ملف الهوك والملفات المسؤولة عن التلويث، وتثبيت الهوك لنتخلص من الفيروس نهائياً.

4. مراحل حذف الفيروسات

يجب احترام الترتيب دائماً لعدم الوقوع في حلقة مفرغة:

1) تنظيف الـــ Process.

2) إلغاء الهوك في حالة وجوده.

3) تنظيف الملفات المسؤولة عن التلويث في السواقات وغيرها.

4) تنظيف الريجستري.

5. الخاتمة

الحمد لله تم هذا الدرس وإن شاء الله تجد فيه المتعة والفائدة

– الملفات التي عمل عليها هذا الدرس مرفقة ليتسنَّ لك أخي الكريم التجربة مع اتباع كل المراحل والوصول إلى النتيجة المبتغاة

الثغرات التي تسهل كسر حماية البرامج المبرمجة بـ VB6

كل ما يتعلق بالدرس ستجده بالمرفقات (التحميل من مرفقات الكتاب).

أغلب من يقومون بحماية برامجهم يعتمدون واحدة أو أكثر من الطرق التالية:

أولاً: كائن غير مفعّل في النسخة التجريبية (زر أو مربع نص أو بند في قائمة وما شابه.).

ويتم ذلك بإحدى الطريقتين :

1) عن طريق الكود، وهو الحل الأسلم.

2) عن طريق تعديل خصائص الكائن، وهذه الطريقة تدل على ضعف خبرة المبرمج بالحماية.

ثانياً: نافذة مزعجة تخرج عند إقلاع البرنامج (Splash)، وهي على نوعين :

1) إما أن تكون MsgBox.

2) وإما أن تكون نافذة يبرمجها المبرمج ثم يستدعيها من داخل البرلمج.

ثالثاً: MsgBox تخرج عند إدخال بيانات تسجيل غير صحيحة Bad Boy.

رابعاً: إغلاق البرنامج لنفسه عند انتهاء المدة التجريبية أو عند إدخال بيانات غير صحيحة

خامساً: مقارنة السريال الحقيقي بالسريال الخاطئ.

هذه كلها ثغرات تسهل على المهندس العكسي كسر البرنامج، وسيتم في هذا الدرس إن شاء الله توضيح هذه الثغرات وكيفية تجاوزها وكسر البرنامج.

I. أولاً: كائن غير مفعّل في النسخة التجريبية

I.I المثال الأول : [زر غير مفعّل]

قم بتشغيل الملف المرفق cmd NEnable ستجد أن زر الحفظ فيه غير مفعّل:



حمّل البرنامج في OllyDbg واذهب إلى قسم الدوال عن طريق ضغط الزر الأيمن واختيار Search For ومن ثم اختيار All intermodular Calls ووضع نقط توقف على الدالة VbaObjSet



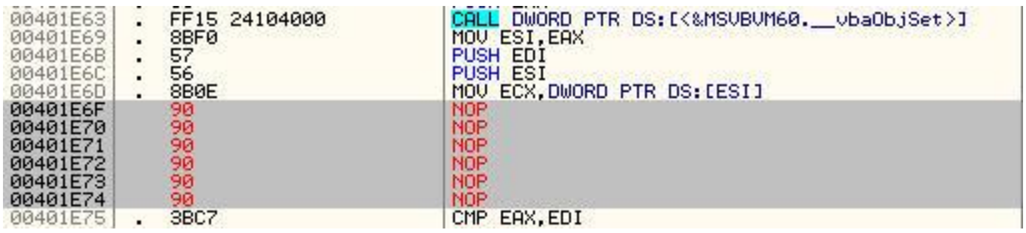
```
00401E63  .  FF15 24104000   CALL DWORD PTR DS:[<&MSVBVM60.__vbaObjSet>]
00401E69  .  8BF0            MOV ESI,EAX
00401E6B  .  57              PUSH EDI
00401E6C  .  56              PUSH ESI
00401E6D  .  8B0E            MOV ECX,DWORD PTR DS:[ESI]
00401E6F  .  FF91 8C000000   CALL DWORD PTR DS:[ECX+8C]
00401E75  .  3BC7            CMP EAX,EDI
```

هذه الدالة مهمة جداً، فهي مسؤولة عن وضع الأداة: من حيث كونها مفعّلة أو غير ذلك، انزل للأسفل حتى تصل للنداء الثاني الذي تراه في الصورة، والذي يحمل القيمة [ECX+8C] والقيمة 8C تشير إلى أن الزر غير مفعّل، ولتفعيله نقوم بأحد الأمرين:

1) عمل NOP لهذا النداء.

2) القفز إلى CMP EAX,EDI.

سنجرب الخيار الأول :



```
00401E63  .  FF15 24104000   CALL DWORD PTR DS:[<&MSVBVM60.__vbaObjSet>]
00401E69  .  8BF0            MOV ESI,EAX
00401E6B  .  57              PUSH EDI
00401E6C  .  56              PUSH ESI
00401E6D  .  8B0E            MOV ECX,DWORD PTR DS:[ESI]
00401E6F     90              NOP
00401E70     90              NOP
00401E71     90              NOP
00401E72     90              NOP
00401E73     90              NOP
00401E74     90              NOP
00401E75  .  3BC7            CMP EAX,EDI
```

تهانينا لقد تم كسر البرنامج وتفعيل الزر.

1.2 المثال الثاني : [نص يدل على أن نسختنا تجريبية وغير مسجلة]

قم بتشغيل الملف المرفق Lbl NEnable وستجد في وسطه عبارة (نسخة غير مسجلة) :



حمّل البرنامج في OllyDbg ثم اذهب إلى قسم الدوال وضع نقط توقف على الدالةVbaObjSet :

```
00401CF3  .  FF15 1C104000   CALL DWORD PTR DS:[<&MSVBVM60.__vbaObjSet>]
00401CF9  .  8BF0            MOV ESI,EAX
00401CFB  .  68 DC154000     PUSH Lbl_NEna.004015DC
00401D00  .  56              PUSH ESI
00401D01  .  8B0E            MOV ECX,DWORD PTR DS:[ESI]
00401D03  .  FF51 54         CALL DWORD PTR DS:[ECX+54]
00401D06  .  3BC7            CMP EAX,EDI
00401D08  .  DBE2            FCLEX
```

قلنا: إنها مهمة جداً، ومسؤولة عن وضع الأداة : من حيث كونها مفعّلة أو غير ذل ك، ومسؤولة عن إظهار نص داخل الأداة . انزل للأسفل حتى تصل للنداء الثاني الذي تراه في الصورة، والذي يحمل القيمة[ECX+54] ولحذف النص نقوم بأحد الأمرين :

1) عمل NOP لهذا النداء.

2) القفز إلى CMP EAX,EDI.

هذه المرة سنجرب الخيار الثاني :

```
00401D03   v EB 01     JMP SHORT Lbl_NEna.00401D06
00401D05     54        PUSH ESP
00401D06  .> 3BC7      CMP EAX,EDI
00401D08  .  DBE2      FCLEX
```

تهانينا لقد تم حذف النص من داخل الأداة.

1.3 المثال الثالث : [بند الحفظ غير مفعّل في القائمة]

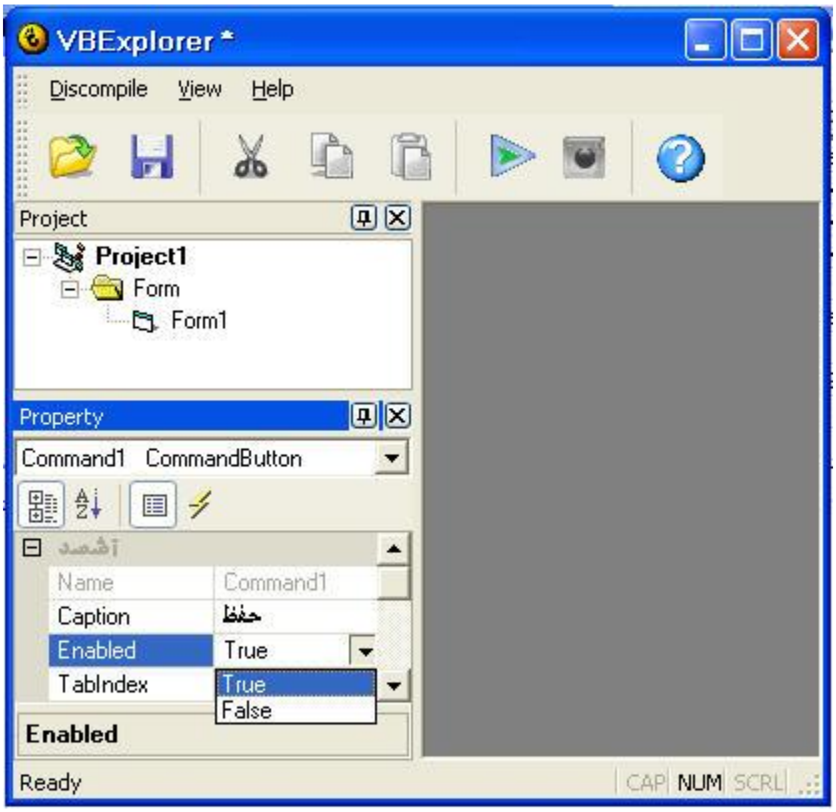
كما فعلنا في المثالين السابقين نفعل مع هذا البند، وعندها ستجد أنه قد تم تفعيل البند

ملاحظة : في الأمثلة السابقة قام المبرمج بإلغاء تفعيل الأدوات عن طريق الكود، وهذا يدل على ذكاءه، ولكن أحياناً يقوم بعض المبرمجين بإلغاء تفعيل الأدوات عن طريق الخصائص :

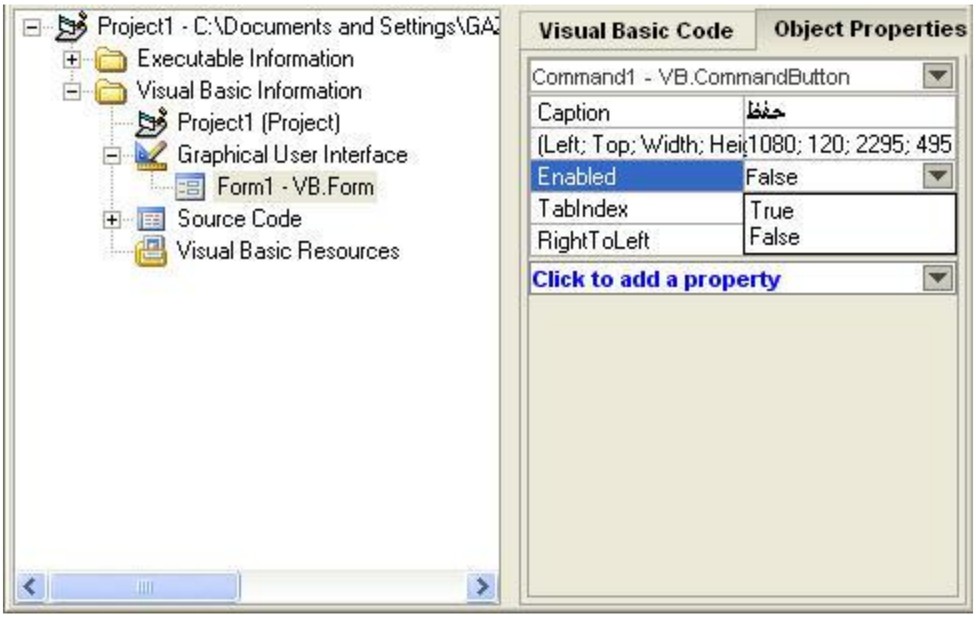


انظر خاصية Enabled = False وهذا خطأ فادح من المبرمج؛ لأن المهندس العكسي سيقوم بكسر البرنامج عن طريق استخدام أحد محررات الفيجوال بيسك؛ مثل: VBExplorer أو VBReFormer ومن ثم سيقوم بتفعيل الكائن.

حمل المرفق Obj NEnable في VBExplorer وقم بتفعيل الزر كما هو موضح في الصورة ثم اضغط الزر حفظ:



أو تستطيع تحميل المرفق Obj NEnable في برنامج VBReFormer ومن ثم تقوم بتفعيل الزر كما هو موضح بالصورة ومن قائمة File تختار Save binary as :



ستجد بعد ذلك أنه قد تم تفعيل الزر، وهكذا تقوم بتفعيل الأدوات وتعديل العبارات كما تشاء

2. ثانياً: نافذة مزعجة تظهر عند إقلاع البرنامج

2.1 المثال الأول: MsgBox تظهر عند إقلاع البرنامج كتنبيه على أنه نسخة غير مسجلة

الـــ MsgBox بوابة تمكّن المهندس العكسي من كسر البرنامج بسهولة.

شغل البرنامج المرفق MsgBox وستجد عند إقلاعه رسالة مزعجة تخبرك بأنه غير مسجل، ولكن كيف ستتخلص منها؟



حمل البرنامج في OllyDbg واضغط F9 للتشغيل، ها هي الرسالة تظهر أمامك، قم بعمل إيقاف للبرنامج بالضغط على F12 لقد توقف البرنامج، الآن اضغط Alt+F9 ثم عد للرسالة واضغط الزر موافق، ستجد أن البرنامج قد توقف عند المكان الذي تُستدعى منه الرسالة :

```
00402160  .  51             PUSH ECX
00402161  .  52             PUSH EDX
00402162  .  FFD7           CALL EDI
00402164  .  50             PUSH EAX
00402165  .  FF15 1C104000  CALL DWORD PTR DS:[<&MSVBVM60.#595>]
0040216B  .  8D85 0CFEFFFF  LEA EAX,DWORD PTR SS:[EBP-1F4]
```

اصعد لبداية الروتين وضع نقطة توقف عنده، ثم أعد تشغيل البرنامج ستجد أنه توقف عند بداية الرويتن:

```
00401B10  > 55             PUSH EBP
00401B11  . 8BEC           MOV EBP,ESP
00401B13  . 83EC 0C        SUB ESP,0C
00401B16  . 68 A6104000    PUSH <JMP.&MSVBVM60.__vbaExceptHandler>
```

الآن قم بتعديل PUSH EBP إلى RETN ثم شغل البرنامج، وستجد أن الرسالة لم تعد تظهر بل سيظهر لك البرنامج مباشرة

2.2. المثال الثاني : مربع حوار يظهر عند الإقلاع كتنبيه على انتهاء مدة استخدام البرنامج

هذه الطريقة أفضل من استخدام MsgBox حيث تصعّب على المهندس العكسي مهمته.

شغل البرنامج المرفق Color ستجد أمامك نافذة مزعجة Splash تخبرك بانتهاء المدة التجريبية :



للتخلص منها حمّل البرنامج في OllyDbg واذهب إلى قسم الدوال وضع نقطة توقف على الدالة vbaNew2 (ستجد لها أكثر من مكان؛ لذا ضع نقطة توقف عليها جميعاً) ثم شغل البرنامج، ستجد أنه قد توقف هنا :

```
004025BC  . 85C0           TEST EAX,EAX
004025BE  .v 75 10         JNZ SHORT Color.004025D0
004025C0  . 68 24404000    PUSH Color.00404024
004025C5  . 68 C0174000    PUSH Color.004017C0
004025CA  . FF15 5C104000  CALL DWORD PTR DS:[<&MSVBVM60.__vbaNew2>]
004025D0  > 83EC 10        SUB ESP,10
```

قم بالتتبع قليلاً ستجد أن النافذة يتم استدعاؤها من هذا النداء:

```
00402605  . 8B45 DC        MOV EAX,DWORD PTR SS:[EBP-24]
00402608  . 8943 04        MOV DWORD PTR DS:[EBX+4],EAX
0040260B  . 894B 08        MOV DWORD PTR DS:[EBX+8],ECX
0040260E  . 8B4D E4        MOV ECX,DWORD PTR SS:[EBP-1C]
00402611  . 894B 0C        MOV DWORD PTR DS:[EBX+C],ECX
00402614  . FF92 B0020000  CALL DWORD PTR DS:[EDX+2B0]
```

لتجاوز النافذة أنت أمام خيارين :

1) عمل NOP للنداء الذي يستدعي النافذة.

2) الصعود إلى بداية الروتين وتعديل PUSH EBP إلى RETN.

عندها ستجد أنه قد تم تجاوز النافذة، وأصبح بإمكانك الدخول للبرنامج.

3. ثالثاً: MsgBox تخرج عند إدخال بيانات تسجيل غير صحيحة Bad Boy

لقد شرحنا فيما سبق كيف يتم التخلص من الـــMsgBox، وهي طريقة يمكن تطبيقها على عدة برامج فلا داعٍ للتكرار.

4. رابعاً: إغلاق البرنامج لنفسه عند انتهاء المدة التجريبية أو عند إدخال بيانات غير صحيحة

شغّل البرنامج المرفق Finsh ستجد أنه سيتم إغلاقه بعد ثلاث ثوانٍ، وهذه ثغرة يمكن التخلص منها بسهولة

حمل البرنامج في OllyDbg واذهب إلى قسم الدوال، وضع نقطة توقف عند الدالة VbaEnd . شغّل البرنامج، وانتظر ثلاث ثوانٍ ستجد أن البرنامج بعدها قد توقف عند الدالة VbaEnd :

```
00401CF6  .  50            PUSH EAX
00401CF7  .  8945 08       MOV [ARG.1],EAX
00401CFA  .  8B10          MOV EDX,DWORD PTR DS:[EAX]
00401CFC  .  FF52 04       CALL DWORD PTR DS:[EDX+4]
00401CFF  .  FF15 08104000 CALL DWORD PTR DS:[<&MSVBVM60.__vbaEnd>]
```

للتغلب على مشكلة إغلاق البرنامج لنفسه فأنت أمام حلين:

1) عمل NOP للنداء المسؤول عن إغلاق البرنامج، والذي تراه أمامك.

2) الصعود إلى بداية الروتين وتعديل PUSH EBP إلى RETN.

وبعدها ستجد أن البرنامج لن يقوم بإغلاق نفسه إطلاقاً.

5. خامساً: مقارنة السريال الحقيقي بالسريال الخاطئ

مقارنة السريال الحقيقي بالسريال الذي يدخله المستخدم ثغرة تمكن المهندس العكسي من قنص السريال الحقيقي

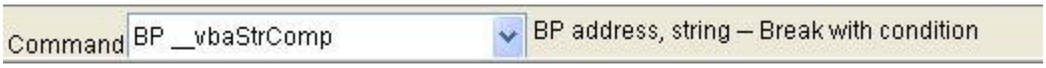
شغل البرنامج المرفق Serial ستجد أنه يطالبك بالتسجيل، وعند إدخالك لأي سريال سيختفي الزر، وستظهر لك عبارة تدل على أن السريال خطأ، وهذا ذكاء من المبرمج؛ لأنه لم يستخدم الـــMsgBox التي تسهل كسر البرنامج.

ولكن كيف يتم كسر البرنامج وقنص السريال الحقيقي؟

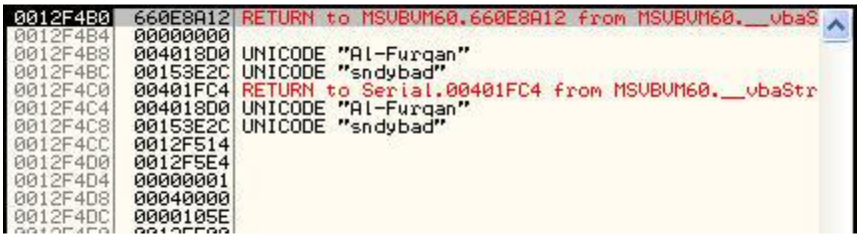
هنالك طريقتان :

5.1 الطريقة الأولى : استخدام OllyDbg

حمّل البرنامج في OllyDbg وقم بتشغيله ثم ضع نقطة توقف BP __vbaStrComp داخل المربع Command ، واضغط من لوحة المفاتيح الزر Enter :



الآن قم بإدخال أي سريال خاطئ؛ مثلاً : sndybad ثم اضغط الزر تسجيل، وعندها ستجد أن البرنامج قد أظهر لك السريال الحقيقي في المكدس عن طريق مقارنته بالسريال الذي أدخلته :



5.2 الطريقة الثانية : استخدام الأداة vbaStrCmp.2.1

عن طريق الأداة السحرية vbaStrCmp.2.1 والتي تقوم بقنص بيانات التسجيل الخاصة بـــ vb6 :

انسخ البرنامج Serial والصقه في مجلد الأداة، ثم قم بتشغيله من داخل مجلد الأداة، وأدخل أي سريال خاطئ، ثم ا ضغط تسجيل وشاهد ما ستفعله الأداة vbaStrCmp.2.1 :



لقد قامت الأداة بقنص السريال الصحيح كما تشاهد في المستطيل الأحمر، بعد مقارنته بالسريال الخاطئ

ما عليك الآن إلا أن تكتب السريال الذي قنصته، وهانينا تم كسر البرنامج وأصبح مسجلاً بشكلٍ كامل





بذلك نكون قد انتهينا من هذا الدرس والحمد لله.

والغاية من الدرس: أن تكون أخي مبرمج vb6 على بينة من حمايتك؛ بحيث تعرف نقاط قوتها وضعفها، وبحيث تقوم باستخدام هذه الطرق كتمويه للمهندس العكسي أو تجد لها مخرجاً يسد هذه الثغرة.

تذكر دائماً أنه لا يوجد برنامج لا يكسر، ولكن يوجد برنامج يصعب كسره، وأقوى الحمايات هي الحماية التي تبتكرها بنات أفكارك وتصنعها أناملك، أما الحمايات المعلّبة والجاهزة، فهي كالوجبات السريعة لا تسمن ولا تغني من جوع

حماية الدونغل Dongle Protection

I. مقدمة

I.1. تعريف الدونغل و أنواعها

الدونغل عبارة عن قطعة فيزيائية (Hardware) يتم حماية البرمجيات بها، و لكي يعمل البرنامج بشكل جيدٍ وكامل يجب أن تكون هذه القطعة متصلة مع الحاسب، و أصبح الآن هنالك ثلاثة أنواع منها:

Parallel و الثانية USB و الثالثة Smart Card وهي الأحدث.

لسنا خبراء إلكترونيات لكي نشرح الدونغل من الداخل، و لكن حسب ما قرأنا فهي عبارة عن ذاكرة، كل ما بداخلها مشفر، ولا يمكن قراءته بحال من الأحوال بدون Keys و يختلف التصميم من شركة مصنعة لأخرى.

حيث تعتمد الشركات على خوارزميات تشفير؛ مثل (RSA و AES...) للقراءة أو الكتابة على الدونغل؛ وذلك لمنع الحصول على محتوياته.

I.2. لمحة عن الشركات المصنعة للدونغل

إذا ما بحثت في الإنترنت فسوف تجد أن شركة (Sentinel) safenet المختصة في نظم الحماية هي التي تتربع على عرش الشركات المصنعة للدونغل ، وهذا موقعها (http://www.safenet-inc.com) . وأحد منتجاتها هو الدونغل، و الذي يعتبر محط انظار جميع الكراكر، و ذلك لشهرته و استخدامه بكثرة من قبل شركات البرمجيات

صورة للدونغل الخاص بهم :



ملاحظة : كان اسم هذه الشركة Rainbow في الأسبق قبل أن تتحول إلى Safenet.

الشركة الثانية هي شركة علاء الدين الإسرائيلية (نسأل الله تعالى أن يأتي اليوم الذي يبيد فيه هذا الكيان الصهيوني)، و المعروفة بـــ HASP ، وهذا موقعها (http://www.aladdin.com/hasp), وهي أيضاً شركة رائدة في هذا المجال، ولكن ليس بقوة الشركة السابقة لا من حيث عدد الزبائن، ولا من حيث قوة الحماية. وهذه عائلتها الجديدة :



هنالك شركات أخرى، منها :

- SmartDongle الأمريكية، وموقعها
- Matrix ألمانية الصنع، وموقعها
- و ه ناك العديد من الشركات الأخرى، ويمكن الحصول على معلومات عنها من خلال صفحة الويكي :

2. الحماية باستخدام الدونغل

مع الأسف أكثر الناس – وخصوصاً في عالمنا العربي– يستخدم الدونغل بطريقة غير صحيحة، مما يؤدي بالنهاية إلى كسر البرنامج المحمي به بثوانٍ؛ حيث أنّ الكثير منهم يعتقد أنه بمجرد استخدام الدونغل في حمايةِ البرنامج فهذا يعني أنّ البرنامج أصبح من الصعب كسره أو لا يمكن كسره نهائياً , مثل هؤلاء كمثل الذي لديه دبابة مجهزة بالعديد من الأ سلحة بداخلها، تستطيع صد الكثير من أنواع الهجمات التي يمكن تنفيذها، ولكنهم يطفئون هذه الدبابة و يجلسون خارجها ، ويكتفون بالقول: إننا نمتلك دبابة فلا يحاول أحد الهجوم علينا؛ لأنه لن يستطيع هزيمتنا !!

النقطة الأضعف في حماية أي برنامج بالدونغل هي دمج أو ربط حماية الدونغل (Dongle) في البرنامج نفسه، وهذا ما يقوم به المبرمج وليس الشركة المصنعة للدونغل (Dongle), فالشركة المصنعة للدونغل تعطيك الأدوات و الأسلحة مع شرح لكل أداة وكيفية الاستخدام الأمثل لها, ولكن ما ذنب الشركة المصنعة إذا لم تقرأ ما أعطتك إياه؟!

أغلب الشركات تلجئ إلى حماية منتجاتها بدونغل عندما يكون منتجها غالي الثمن، ولكن غالباً ما يكون المبرمجون داخل هذه الشركات يجهلون تماماً مدى قوة الكراكر و الأدوات التي معه, يجهلون تماماً أن لديه debug tool بحيث تمكنه من المشي على الكود الخاص بهم خطوة بخطوة, يجهلون شيئاً آخراً اسمه التشفير، وأنه يمكن استخدامه لحماية التطبيقات أيضاً وليس فقط البيانات.

في الحقيقة إن كل الكراكر متفقون على أنّ الحماية باللونغل هي من أصعب الحمايات, هذا فيما إذا تم استخدامها بالطريقة الصحيحة، وإليك بعض النصائح الهامة التي يجب عليك اتباعها عندما تريد أن تقوم باستخدامهذه الحماية :

3. نصائح لحماية تطبيقك بشكل جيد باستخدام الدونغل Dongle

3.1 قراءة ال Manuals والاطلاع على ال SDKs

على المبرمج أن يقرأ الـ Manuals أو الـ SDK التي تعطى مع الد ونغل, و هي أهم خطوة في طريقك لحماية الد ونغل بشكلٍ صحيح, وقد رأيت الكثير من الناس الذين يستخدمون حماية الدونغل دون قراءة سطر واحد في الـ Manual الذي يأتي مع الدونغل, ولعلهم يستصعبون الأمر و يجدونه معقداً؛ بحيث يبعدهم عن قرا ءته. حيث أنّ التعامل مع الدونغل ليس Standard بل يختلف من شركة مصنعة لشركة أخرى. و كل شركة لها توابعها و مكتباتها الخاصة ويجب الاطلاع عليها كلها, بل و تمحيصها.

3.2 بعثر توابع استدعاء الدونغل (Distribute Function Calls)

قم بتوزيع الـ Function calls الخاصة بالدونغل في برنامجك، و ليس فقط استدعاءها عند إقلاع البرنامج, طبعاً هذا سوف يصعب على الكراكر المهمة؛ فإزالة عدة استدعاءات مبعثرة في التطبيق أصعب بكثير من استدعاء واحد يتم عند إقلاع التطبيق. يجب أن يكون الاستدعاء الأول عند بداية البرنامج فقط هو بوابة المتاهة التي سوف يدوخ بها الكراكر . أيضاً اجعل الفحص يتم في أغلب العمليات الهامة التي يقوم بها البرنامج؛ مثل الطباعة وعمل الحسابات الهامة وغير ذلك.

3.3 فحص متكرر (Frequent Check)

يجب أن تقوم بعمل تفحص لوجود الد ونغل بشكل متكرر , أي أن تقوم بفحص وجوده كل دقيقة مثلاً , فليس من المعقول أن يتم الفحص فقط عند بداية فتح البرنامج؛ فهذا سوف يمكن المستخدم من استخدام عدة نسخ كيف؟؟!

لو قام المستخدم بتنصيب عدة نسخ على عدة حواسيب متقاربة من بعضها, ولو فرضنا أنك تقوم بفحص وجود اللونغل فقط عند بدء التشغيل, سوف يضع الدونغل في الحاسب الأول و يشغل البرنامج فيعمل, و من ثم سريضعه في الحاسب الثاني و من ثم يشغل البرنامج فيعمل و هكذا.. ولو كان هناك فحص متكرر لوجود الدونغل مثلاً باستخدام Timer لما تمكن المستخدم من فعل هذا, كما ويمكنه عمل Hibernate للأجهزة لتبقى البرامج شغالة.

3.4. استخدام ذاكرة الدونغل

- استخدم ذاكرة الدونغل في أمور تجعل من عمل البرنامج بدونها مستحيلا ؛ مثلاً جزء من الكود هام جداً, مثلاً يكون هذا الكود هو المسؤول عن إقلاع البرنامج, و يتم جلبه من اللونغل و إضافته إلى البرنامج عند بدء التشغيل
- بعض البارامترات والمتحولات الهامة لعمل البرنامج و يتم جلبها أيضاً من الدونغل في كل مرة تحتاجها.
- إذا كان لديك خوارزميات تشفير ما تستخدمها في البرنامج فضع الـ Keys الخاصة بها داخل ذاكرة الدونغل.
- بعض الأرقام العشوائية في الذاكرة والغير مستخدمة تقوم بفحص وجودها وتطابقها من البرنامج
- يمكنك أيضاً فتح ميزات أو سماحيات في برنامجك بناء على ذاكرة الدونغل ؛ ولكل ميزة ثمنها. ولتوضيح الفكرة أكثر: لو فرضنا أن البرنامج الذي تقوم بعمله يقوم بالقراءة و الكتابة من ملف , فيمكن عمل نسختين من البرنامج : الأولى للقراءة فقط, و الثانية للقراءة والكتابة. و كل نسخة لها ثمنها.
-

3.5. لا تستخدم شروط فحص بسيطة

بعد قراءة قيم المتحولات من ذاكرة الدونغل استخدم شروط فحص معقدة بعض الشيء عوضاً عن شروط الفحص العاديةP على سبيل المثال: إذا أردت فحص قيمة متحول تساوي 255 فلا تقم بفحصها بهذا الشكل :

كود:

```
If (MyVar = 255)....
```

و إنما حاول استخدام صيغة معقدة أكثر؛ مثلاً :

كود:

```
If ((MyVar + 90)/7 = SQRT (MyVar)*3)....
```

صدق أن الـ Cracker سوف يمضي الكثير من الوقت لفهم هذه الصيغة، و خصوصاً أنّ الكود الذي يراه منخفض المستوى (أسمبلي).

3.6. استخدام الKEY الخاص بالدونغل

أغلب هذه الحمايات تأتي اليوم بحيث تستطيع وضع 128 bit key بداخلها , و يمكن عمل الكثير من الأ مور باستخدام خاصية التشفير وفك التشفير؛ مثلاُ نقوم بتشفير بعض القيم باستخدام البرنامج الذي يأتي لبرمجة الدونغل , وتخزين هذه القيم في التطبيق مشفرة,

بعدها نقوم بإرسال النص الغير مشفر إلى الدونغل ليقوم بتشفيره باستخدام الـ Key و يعيد النتائج لنقوم بمقارنته مع القيم المشفرة. فإذا كانت متطابقة نقوم بتشغيل البرنامج و إلا نقوم بإلاقه.

3.7. استخدم الحماية الخاصة بالشركة أو ما يعرف بال Shell

دائماً يأتي برنامج من الشركة يقوم بحماية البرنامج بشكل تلقائي ، وحتى بدون كتابة أي سطر كود, فقط يطلب منك البرنامج لكي يقوم بحمايته, من الافضل استخدام هذه الحماية فوق الحماية التي تقوم بها أنت، فهم يقومون على الأغلب بتشفير الكود داخل التطبيق حسب الـ 128 bit key المخزن داخل الدونغل، و يتم فك التشفير عند بدء تشغيل البرنامج, و يضعون الكثير من الحمايات و العقبات التي تعيق تقدم الكراكر في كسر البرنامج و هي مفيدة جداً.

مثلا يقدمون حمايات كالتالي:

- Compress sections.
- Merge sections.
- Protect Sections.
- Force Trim Sections.
- Quit if debugged.

3.8. الحماية من ال Emulator

ما هو الـ Emulator ؟

هو عبارة عن برنامج software يحاول محاكاة، أي تقليد عمل الدونغل، بحيث يعمل البرنامج بدون وجود الدونغل الأصلي، و إنما باستخدام هذا الـ Software حيث يعتمد مبدأ عمله على التنصت على ال Port الخاص بالدونغل ليراقب جميع المخرجات و المدخلات على هذا الـ Port و بعدها ليقوم بتقليدها. أصبحت هذه البرامج اليوم كثيرة الانتشار, ولكن لها دواء ناجع و شافٍ, وهو توليد أرقام عشوائية باستخدام التابع Random مثلاً و إرسالها إلى الدونغل ليتم تشفيرها، و من ثم نقوم بإرسال القيم المعادة (المشفرة) لكي يتم فك تشفيرها، و من ثم نقارنها مع القيمة البدائية التي تم توليدها بواسطة التابع Random .

مثال توضيحي :

لو فرضنا أنه تم إعادة الرقم 364732 من خلال التابع Random بعد إرساله إلى الدونغل ليقوم بتشفيره (Encrypt) باستخدام 128 bit key حصلنا على القيمة 894837199 ثم نقوم بإعادة إرساله إلى الدونغل ليقوم بفك تشفيرها (Decrypt) فإذا حصلنا على القيمة 364732 نتأكد أن الدونغل موجود، وأنّ القيم لم يتم إعادتها باستخدام Emulator حيث أنّ الـ Emulator يعتمد على التسجيل، و بما أننا نستخدم التابع Random فسيكون التوليد عشوائياً، فماذا سوف يسجل الـ Emulator ؟!

ملاحظة: جميع هذه الحمايات تحتوي على توابع تشفير و فك التشفير لقيم نصية ترسلها إلى الدونغل، و يتم هذا التشفير باستخدام خوارزميات تشفير معقدة (AES) باستخدام مفتاح تشفير 128 bit.

في النهاية: إذا اتبعت النصائح الموجود أعلاه فإن شاء الله سوف يصبح من المستحيل كسر برنامجك بدون حصول الـ Cracker على الدونغل، و أغلب إن لم يكن 99% من ال Crackers ليس لديهم ثمن الدونغل الأصلي، و لن يقوموا بشرائه كي يحاولوا كسر برنامجك، و حتى لو استطاع أحدهم الحصول عليه سصبح المشوار بعيد اً جداً على الـ Cracker وصدق سوف يقضي الكثير من الوقت لكي يستطيع كسره, و خصوصاً إذا وضعت بعض الفلفل و الملح على الطبيق؛ بحيث لا يستطيع تذوقه بالشكل المناسب.

طبعاً إذا وقع دونغل ما بيد كراكر ، فنقطة الضعف سوف تكون بعد تشغيل البرنامج ، فسوف يكون هناك نسخة غير مشفرة من البرنامج في الذاكرة، و بعدها يستطيع عمل Dump و من ثم استخلاص نسخة غير مشفرة ، و بعدها يستطيع التعامل معها كما هو الحال في البرنامج الذي قمت بكسره

مطورو شركة الدونغل اكتشفوا هذا الشيء؛ لذلك تراهم يقسمون البرنامج إلى قطع كثيرة، و لكنها مشفرة، و تبقى قطعة واحدة غير مشفرة، و هي التي تقوم بفك تشفير أي قطعة من هذه القطع حسب الطلب . و سيكون من الصعب جداً على القرصان تجميع هذه القطع و استخلاص برنامج غير مشفر.

و هذا كله يتم باستخدام أداة الحماية التي يوفرونها معالدونغل؛ لذلك من الضروري الاعتماد عليها فوق الحماية التي تقوم بها

4. مساوئ استخدام الدونغل في حماية البرامج مقارنة بالوسائل الأخرى

تكلفة إضافية تضاف إلى سعر البرنامج, فسعرها تقريبا بين 20- 40 دولاراً؛ لذلك يتم استخدام هذا النوع من الحمايات في البرامج المرتفعة السعر.

بما أنها قطعة Hardware فهي قابلة للتلف كباقي القطع، و لكن الشركات المصنعة عادة ما تقدم كفالة لها

صعوبة إيصالها إلى المستخدمين ، فيجب شحن الدونغل بواسطة شركات الشحن , وإنما في الطرق الاخرى يمكن عمل مولد لأرقام التسجيل من خلال موقع إنترنت.

قابلة للسرقة من على الحاسب ، وهذه حدثت مع كثير من الشركات التي نعرفها، وبالتالي خسارة النسخة؛ لأنّ شركة البرمجيات لن تعطيك دونغل أخرى بسهولة.

صعوبة إدارتها مقارنة بالطرق الأخرى, فغالباً ما تطلب موظفاًمسؤولاً عنها.

تعتمد في قوتها على قوة وصلابة الشركة المصنعة, وفي حال التمكن من كسر الدونغل فيزيائياً أو التمكن من اختراق الـ SDK الخاصة بالشركة فهذا يعني انتهاء حماية برنامجك بالكامل.

5. مواضيع ذات صلة بالموضوع من المنتدى :

- http://www.at4re.com/f/showthread.php?t=2787
- http://www.at4re.com/f/showthread.php?t=2783

الخاتمة

﴿ وَأَنزَلَ اللَّهُ عَلَيْكَ الْكِتَابَ وَالْحِكْمَةَ وَعَلَّمَكَ مَا لَمْ تَكُن تَعْلَمُ وَكَانَ فَضْلُ اللَّهِ عَلَيْكَ عَظِيمًا ﴾

[النساء : 113]

وأخيراً وليس آخراً .. نصل معكم إلى نهاية المطاف بعد أن أبحرنا بكم في دروس متنوعة من علم الهندسة العكسية وبأدق تفاصيلها.

فبعد أن كان المستخدم العربي يبحث في الشبكة العنكبوتية عن أي شيء حول الهندسة العكسية وبعض فروعها ثم لا يجد إلا درساً متواضعاً أو مقالة مجتزأة، أصبح بإمكانه اليوم أن يقرأ الكثير حولها، وهذا الأمر لم يكن عن فراغ أيها القارئ الكريم، بل كان أعضاء الفريق يعملون بجد ونشاطٍ ودأبٍ دون جزاء أو شكور، وستلمس ذلك من خلال ما قدمناه لك في الإصدار الأول من كتابنا [مدخل إلى الهندسة العكسية]، ومن خلال ما طرحناه في الإصدار الثاني [الهندسة العكسية : خطوة إلى الأمام]، ومن خلال ما نفكر بطرحه في الإصدار الثالث إن شاء الله.

نأمل أن يحقق هذا الكتاب ما نصبو إليه من إيصال هذا العلم للجميع، ورفد المبرمجين بالطرق والوسائل التي تجعلهم قادرين على التعامل مع برمجياتهم بشكلٍ أدق وأشمل.

اللهم لك الحمد كله ولك الشكر كله وإليك يرجع الأمر كله، والصلاة والسلام على خير الأنام وعلى آله وصحبه وسلم.

وآخر دعوانا أنِ الحمد لله رب العالمين...

الفريق العربي للهندسة العكسية

Arab Team 4 Reverse Engineerin

INDEX


at4re
ARAB TEAM 4 REVERSE ENGINEERING
Arab Team
Reverse Engineering
1430